Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 225 (2009) 181–194

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Properties of General Semantic Operators Determined by Logic-Based Systems

Máire Lane [1,2]

Department of Mathematics
University College Cork
Cork, Ireland

Anthony Karel Seda[1,3]

Department of Mathematics
University College Cork
Cork, Ireland

Abstract

We discuss a very general semantic operator arising within logic-based programming systems from an algebraic point of view, and show how it connects four interesting aspects of computation: neural networks, conventional logic programming, constraint logic programming, and simple models of uncertainty in logic-based systems.

Keywords: Semantic operators, (constraint) logic programming, neural networks, semirings, uncertain reasoning.

1 Introduction

A number of semantic operators have been defined and studied in the literature on the semantics of logic programs P. These operators map interpretations of P to interpretations of P, and the usual goal in defining them is to realize the various well-known semantics for P (such as the supported models, the perfect models, the stable models, the well-founded models etc.) as fixed points of one or other of these operators. A fairly general operator Ψ_P which subsumed certain of these semantics was defined by Fitting in [5] over Kleene's three-valued logic and Belnap's four-valued logic $\mathcal{FOUR}$.

[1] The authors thank the Boole Centre for Research in Informatics at University College Cork for substantial support during the preparation of this paper.

[2] Email: maireln@eircom.net

[3] Email: a.seda@ucc.ie

doi:10.1016/j.entcs.2008.12.073

A quite general extension $\mathfrak{T}_P$ of Fitting's operator was introduced and studied in [10], [11] and [9] over logics $\mathcal{T}$ satisfying certain conditions. This was done with a view to giving a unified treatment of several of the various semantics mentioned above, and to extending the ideas of logic programming beyond its usual boundaries, this latter point being the main point we take up here. Furthermore, the issue of the computation of this operator by means of artificial neural networks was addressed in [9] in an effort to give logical semantics to neural networks, and more generally to nature-inspired models of computation taking neural networks as a primary and motivating example. The key idea underlying the definition and properties of $\mathfrak{T}_P$ is that the connectives $\vee$ and $\wedge$ in $\mathcal{T}$ should be *finitely determined* as defined in [11], see Definition 2.1 below. This condition on $\vee$ and $\wedge$ implies that they are idempotent, associative and commutative. Indeed, if $\mathcal{T}$ is finite, then being finitely determined is equivalent to $\vee$ and $\wedge$ being idempotent, associative and commutative; we summarize in Theorem 2.3 for the reader's convenience the basic properties of completely general finitely-determined binary operations $\odot$.

However, there are some extra-logical issues surrounding this operator by virtue of the generality of our definition, and this implies a meeting of certain seemingly disparate ideas. In fact, we give here an abstract definition $\mathfrak{T}_{P,\mathfrak{C}}$ of $\mathfrak{T}_P$ over sets $\mathfrak{C}$ carrying two binary operations, and our definition is general enough for $\mathfrak{C}$ to apply not only to conventional truth sets, but to semirings and *c*-semirings as well. Indeed, when $\mathfrak{C}$ is a *c*-semiring we recover the semiring framework of Bistarelli et al. [1] for studying constraint logic programming. Hence, one can easily apply the results of [9] to the case of constraint logic programs. Furthermore, the extended syntax we employ here allows one to take truth values as literals in the bodies of clauses. Hence, another example of our current philosophy is that we can easily incorporate within our setting Stamate's rather simple framework [12] for handling uncertainty in databases.

Thus, in summary, the objective of this paper is to (1) formulate an abstract, general definition of a semantic operator $\mathfrak{T}_{P,\mathfrak{C}}$, and (2) to show how one can use it, with suitable choices of $\mathfrak{C}$, to consider the interaction between logic programs, constraint logic programs, Stamate's model of uncertainty, and neural computation by means of the results of [9].

The overall structure of the paper is as follows. In Section 2, we discuss finitely-determined operations in general and show how they are used in giving the definition of our general semantic operator $\mathfrak{T}_{P,\mathfrak{C}}$. In Section 3, we show how this operator subsumes the three special cases mentioned above. In Section 4, we discuss monotonicity properties of $\mathfrak{T}_{P,\mathfrak{C}}$ and its Scott continuity. In Section 5, we briefly discuss the question of the computation of $\mathfrak{T}_{P,\mathfrak{C}}$ by artificial neural networks. Finally, in Section 6, we present our conclusions.

2 The General Semantic Operator $\mathfrak{T}_{P,\mathfrak{C}}$

2.1 Finitely-Determined Operations

Let $\mathfrak{C}$ denote a set endowed with binary operations $+$ and $\times$, and with a unary operation $\neg$ satisfying $\neg(\neg c) = c$ for all $c \in \mathfrak{C}$. In [11], the notion of *finitely-determined disjunctions and conjunctions* ($\vee$ and $\wedge$) was introduced for such sets $\mathfrak{C}$ of truth values. We begin by giving this definition for a general binary operation $\odot$ on $\mathfrak{C}$. Note that we assume that $\odot$ has meaningfully been extended to include products $\bigodot_{i\in M} c_i$ of countably infinite families M of elements c_i of $\mathfrak{C}$. Indeed, the way in which we carry out this extension is the main point of the next definition and the discussion following it.

Definition 2.1 Suppose that $\mathfrak{C}$ is a set equipped with a binary operation $\odot$. We say that *products (relative to $\odot$) are finitely determined in* $\mathfrak{C}$ if for each $c \in \mathfrak{C}$ there exists a countable (possibly infinite) collection $\{(R_c^n, E_c^n) \mid n \in \mathcal{J}\}$ of pairs of sets $R_c^n \subseteq \mathfrak{C}$ and $E_c^n \subseteq \mathfrak{C}$, where each R_c^n is finite, such that a countable (possibly infinite) product $\bigodot_{i\in M} c_i$ is equal to c if and only if for some $n \in \mathcal{J}$ we have
(i) $R_c^n \subseteq \{c_i \mid i \in M\}$, and
(ii) for all $i \in M$, $c_i \notin E_c^n$, that is, $\{c_i \mid i \in M\} \subseteq (E_c^n)^{co}$, where $(E_c^n)^{co}$ denotes the complement of the set E_c^n.

We call the elements of E_c^n *excluded values*, we call the elements of $A_c^n = (E_c^n)^{co}$ *allowable values*, and in particular we call the elements of R_c^n *required values*; note that for each $n \in \mathcal{J}$ we have $R_c^n \subseteq A_c^n$, so that each required value is also an allowable value (but not conversely). More generally, given $c \in \mathfrak{C}$, we call $s \in \mathfrak{C}$ an *excluded value for c* if no product $\bigodot_{i\in M} c_i$ with $\bigodot_{i\in M} c_i = c$ contains s, that is, in any product $\bigodot_{i\in M} c_i$ whose value is equal to c we have $c_i = s$ for no $i \in M$. We let E_c denote the set of all excluded values for c, and let A_c denote the complement $(E_c)^{co}$ of E_c and call it the set of *allowable values.*

The following example shows the thinking behind the previous definition; it was originally motivated by the results of [6].

Example 2.2 Consider Belnap's well-known four-valued logic with set $\mathfrak{C} = \mathcal{FOUR} = \{t, u, b, f\}$ of truth values and connectives as defined in Table 1, where t denotes *true*, u denotes *undefined* or *none*, b denotes *both* (true and false), and f denotes *false.*

Taking $\odot$ to be disjunction $\vee$, the sets E and R are as follows.
(a) For t, we have that n takes values 1 and 2, $E_t = \emptyset$, $R_t^1 = \{t\}$, and $R_t^2 = \{u, b\}$.
(b) For u, we have $n = 1$, $E_u = \{t, b\}$ and $R_u = \{u\}$.
(c) For b, we have $n = 1$, $E_b = \{t, u\}$ and $R_b = \{b\}$.
(d) For f, we have $n = 1$, $E_f = \{t, u, b\}$ and $R_f = \{f\}$.

Thus, a countable disjunction $\bigvee_{i\in M} c_i$ takes value t if and only if either (i) at least one of the c_i is equal to t or (ii) at least one of the c_i takes value u and at least one takes value b; no truth value is excluded. As another example, $\bigvee_{i\in M} c_i$ takes value u if and only if at least one of the c_i is u, none are equal to t and none are equal to b. □

Table 1
Truth table for the logic $\mathcal{FOUR}$

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

It turns out that the connectives in all the logics commonly encountered in logic programming, and indeed in many others, satisfy Definition 2.1, and it will be convenient to state next the main facts we need concerning arbitrary finitely-determined operations (see [11] for all proofs).

Theorem 2.3 *Suppose that $\odot$ is a binary operation defined on a set $\mathfrak{C}$. Then the following statements hold.*

(i) *If products relative to $\odot$ are finitely determined in $\mathfrak{C}$, then the operation $\odot$ is idempotent, commutative and associative.*

(ii) *Suppose that products relative to $\odot$ are finitely determined in $\mathfrak{C}$ and that $\mathfrak{C}$ contains finitely many elements $\{c_1, \ldots, c_n\}$. Then, for any collection $\{s_i \mid i \in M\}$, where each of the $s_i \in \mathfrak{C}$ and M is a denumerable set, the sequence $s_1, s_1 \odot s_2, s_1 \odot s_2 \odot s_3, \ldots$ is eventually constant with value s, say. Therefore, setting $\bigodot_{i \in M} s_i = s$ gives each countably infinite product in $\mathfrak{C}$ a well-defined meaning which extends the usual meaning of finite products.*

(iii) *Suppose that products are finitely determined in $\mathfrak{C}$ and that $\bigodot_{i \in M} s_i = c$, where M is a countable set. Then the sequence $s_1, s_1 \odot s_2, s_1 \odot s_2 \odot s_3, \ldots$ is eventually constant with value c.*

(iv) *Suppose that $\mathfrak{C}$ is a countable set and $\odot$ is idempotent, commutative and associative. Suppose further that, for any set $\{s_i \mid i \in M\}$ of elements of $\mathfrak{C}$ where M is countable, the sequence $s_1, s_1 \odot s_2, s_1 \odot s_2 \odot s_3, \ldots$ is eventually constant. Then all products in $\mathfrak{C}$ are (well-defined and are) finitely determined.*

(v) *Suppose that $\mathfrak{C}$ is finite. Then $\odot$ is finitely determined if and only if it is idempotent, associative and commutative.*

Furthermore, for finitely-determined operations $+$ and $\times$ we define partial orders $\leq_+$ and $\leq_\times$ on $\mathfrak{C}$ by $s \leq_+ t$ iff $s + t = t$, and $s \leq_\times t$ iff $s \times t = t$. Note that these orderings are dual to each other if and only if the absorption law holds for $+$ and $\times$, in which case $(\mathfrak{C}, \leq_+, \leq_\times)$ is a lattice. Notice also that because $+$ and $\times$ are finitely determined, $\sum_{c \in \mathfrak{C}} c \in \mathfrak{C}$ is the top element of $\mathfrak{C}$ relative to $\leq_+$, and $\prod_{c \in \mathfrak{C}} c \in \mathfrak{C}$ is the top element of $\mathfrak{C}$ relative to $\leq_\times$. Note, however, that it does not follow that we have bottom elements for these orderings. We further suppose that two elements $\bar{c}$ and $\underline{c}$ of $\mathfrak{C}$ are distinguished, and these elements will be made use of in Section 2.2 and in Section 4. (In some, but not all, situations when $\mathfrak{C}$ is a logic, $\bar{c}$ is taken to be *true*, and $\underline{c}$ is taken to be *false*.)

Notice that finitely-determined operations $+$ and $\times$ need not satisfy the distributive laws. In any event, throughout what follows, $\mathfrak{C}$ will denote a set endowed with binary operations $+$ and $\times$, and $+$ at least will be supposed to be finitely determined and $\times$ will be assumed to be associative for simplicity.

Of particular interest to us are the following three cases.

(1) $\mathfrak{C}$ is a set of truth values, $+$ is disjunction $\vee$ and $\times$ is conjunction $\wedge$.

(2) $\mathfrak{C}$ is a c-semiring (constraint-based semiring) as considered in [1]. Thus, $\mathfrak{C}$ is a semiring, where the top element in the order $\leq_\times$ is the identity element $\mathbf{0}$ for $+$, and the top element in the order $\leq_+$ is the identity element $\mathbf{1}$ for $\times$. In addition, $+$ is idempotent, $\times$ is commutative, and $\mathbf{1}$ annihilates $\mathfrak{C}$ relative to $+$, that is, $\mathbf{1} + c = c + \mathbf{1} = \mathbf{1}$ for all elements $c \in \mathfrak{C}$.

(3) $\mathfrak{C}$ is the set L_m of truth values considered in Section 3, $+$ is max and $\times$ is min.

In fact, it transpires that our main definition (but not all our results) can be made simply in the context of the set $\mathfrak{C}$ with sufficient completeness properties, namely, that arbitrary countable sums can be defined. Indeed, we next turn to making our main definition (of the operator $\mathfrak{T}_{P,\mathfrak{C}}$).

2.2 The operator $\mathfrak{T}_{P,\mathfrak{C}}$

Let $\mathcal{L}$ be a first-order language, see [8] for notation and undefined terms relating to conventional logic programming, and let the set $\mathfrak{C}$ be given, as above. By a $\mathfrak{C}$-*normal logic program P* or a *normal logic program P defined over* $\mathfrak{C}$ we mean a finite set of clauses or rules of the type $A \leftarrow L_1, \ldots, L_n$ (n may be 0, by the usual abuse of notation), where A is an atom in $\mathcal{L}$ and the L_j, for $1 \leq j \leq n$, are either literals in $\mathcal{L}$

or are elements of $\mathfrak{C}$. By a $\mathfrak{C}$-*interpretation* or just *interpretation* I for P we mean a mapping $I : B_P \to \mathfrak{C}$, where B_P denotes the Herbrand base for P. We immediately extend I to negated atoms $\neg A$ by $I(\neg A) = \neg I(A)$, and to $B_P \cup \neg \cdot B_P \cup \mathfrak{C}$ by setting $I(c) = c$ for all $c \in \mathfrak{C}$. (The usual overloading of the symbol $\neg$ will not cause any confusion.) Finally, we let $I_{P,\mathfrak{C}}$ or simply I_P denote the set of all $\mathfrak{C}$-interpretations for P ordered by $\sqsubseteq_+$, that is, by the pointwise ordering relative to $\leq_+$. Notice that the value $I(L_1, \ldots, L_n)$ of I on any clause body is uniquely determined.

To define the *semantic operator* $\mathfrak{T}_{P,\mathfrak{C}}$, we essentially follow [5], allowing for our extra generality, in first defining the sets P^* and P^{**} associated with P. To define P^*, we first put in P^* all ground instances of clauses of P whose bodies are non-empty. Second, if a clause $A \leftarrow$ with empty body occurs in P, add $A \leftarrow \overline{c}$ to P^*. Finally, if the ground atom A is not yet the head of any member of P^*, add $A \leftarrow \underline{c}$ to P^*. To define P^{**}, we note that there may be many, even denumerably many, elements $A \leftarrow C_1$, $A \leftarrow C_2$, ... of P^* having the same head A. We replace them with $A \leftarrow C_1 + C_2 + \ldots$, where $C_1 + C_2 + \ldots$ is to be thought of as a formal sum. Doing this for each A gives us the set P^{**}. Now, each ground atom A is the *head* of exactly one element $A \leftarrow C_1 + C_2 + \ldots$ of P^{**}, and it is common practice to work with P^{**} in place of P. Indeed, $A \leftarrow C_1 + C_2 + \ldots$ may be written $A \leftarrow \sum_i C_i$ and referred to as a (or as the) *pseudo-clause* with *head* A and *body* $\sum_i C_i$.

Definition 2.4 Let P be a $\mathfrak{C}$-normal logic program. We define $\mathfrak{T}_{P,\mathfrak{C}} : I_{P,\mathfrak{C}} \to I_{P,\mathfrak{C}}$ as follows. For any $I \in I_{P,\mathfrak{C}}$ and $A \in B_P$, we set

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

where $A \leftarrow \sum_i C_i$ is the unique pseudo-clause in P^{**} whose head is A. Note that when $\mathfrak{C}$ is understood, we may denote $\mathfrak{T}_{P,\mathfrak{C}}$ simply by $\mathfrak{T}_P$.

We note that $I(\sum_i C_i) = \sum_i I(C_i)$ is well-defined in $\mathfrak{C}$ by Theorem 2.3.

3 Some Special Cases

As mentioned in the introduction to the paper, the operator $\mathfrak{T}_{P,\mathfrak{C}}$ subsumes a number of important cases, and we show next how each of them can be recovered simply by choosing $\mathfrak{C}$ suitably.

3.1 The Standard Semantics of Logic Programming

By choosing $\mathfrak{C}$ to be an appropriate logic, one recovers the standard semantics of conventional logic programs P as fixed points of $\mathfrak{T}_{P,\mathfrak{C}}$. For example, on choosing $\mathfrak{C}$ to be classical two-valued logic, Kleene's strong three-valued logic, and $\mathcal{FOUR}$ one recovers respectively the usual single-step operator T_P, Fitting's three-valued operator Φ_P, and Fitting's four-valued operator Ψ_P, see [5]. Hence, one recovers the associated semantics as the least fixed points of $\mathfrak{T}_{P,\mathfrak{C}}$.

Furthermore, in [13], Wendt studied the fixpoint completion, fix(P), of a normal logic program P introduced by Dung and Kanchanasut in [3]. The fixpoint completion is a normal logic program in which all body literals are negated, and is obtained by a complete unfolding of the recursion through positive literals in the clauses of a program. In fact, Wendt obtained interesting connections between various semantic operators by means of $\mathrm{fix}(P)$. Specifically, he showed that for any normal logic program P, we have (i) $GL_P(I) = T_{\mathrm{fix}(P)}(I)$ for any two-valued interpretation I, and (ii) $\overline{\Psi}_P(I) = \Phi_{\mathrm{fix}(P)}(I)$ for any three-valued interpretation I, where GL_P is the well-known operator of Gelfond and Lifschitz used in defining the stable-model semantics, and $\overline{\Psi}_P$ is the operator used in [2] to characterize the well-founded semantics of P. These connections have the following immediate consequence that GL_P and $\overline{\Psi}_P$ can be seen as special cases of $\mathfrak{T}_{P,\mathfrak{C}}$, and hence that the well-founded and stable-model semantics can be viewed as special cases of the fixed points of $\mathfrak{T}_{P,\mathfrak{C}}$.

3.2 Constraint Satisfaction Problems and Constraint Logic Programs

A constraint satisfaction problem (CSP) is defined over a constraint system $CS = (\mathcal{S}, \mathbb{D}, \mathcal{V})$, where $\mathcal{S}$ is a c-semiring, $\mathbb{D}$ is a finite set called the domain of constraints and $\mathcal{V}$ is a set of variables. A constraint (def, c) consists of a subset $c \subseteq \mathcal{V}$ and a mapping $\mathit{def} : \mathbb{D}^k \to \mathcal{S}$ which assigns a semiring value $a \in \mathcal{S}$ to any k-tuple of elements of $\mathbb{D}$, where k is the cardinality of c. A constraint satisfaction problem (C, var) consists of a set C of constraints and a set $\mathit{var} \subseteq \mathcal{V}$. Thus, we have a set of variables to be assigned to elements of $\mathbb{D}$ in such a way that a set of constraints is to be satisfied. One particularly interesting class of such problems is the class of Semiring Based Constraint Satisfaction Problems. In that particular framework, different constraint systems are chosen by selecting an appropriate semiring. For example:

(i) For classical constraints: $\mathcal{S} = (\{0,1\}, \vee, \wedge, 0, 1)$.

(ii) For fuzzy constraints: $\mathcal{S} = ([0,1], max, min, 0, 1)$.

(iii) For probabilistic constraints: $\mathcal{S} = ([0,1], max, \times, 0, 1)$.

(iv) For weighted constraints: $\mathcal{S} = (\mathbb{R}^+ \cup +\infty, min, +, \infty, 0)$.

(v) For set-based constraints: $\mathcal{S} = (p(A), \cup, \cap, \emptyset, A)$.

In this list, the semiring (i) is just classical two-valued logic and is finitely determined, but (ii), (iii) and (iv) are not finitely determined. If we take a finite number of truth values and use approximations based on a finite number of intervals, then the semiring (ii) is a finitely-determined logic, and semirings (iii) and (iv) have finitely-determined disjunctions. The semiring (v) depends on the cardinality of A: if A is finite then we have a finitely-determined logic, otherwise not.

A standard constraint logic program, see [7], consists of a finite set of clauses of the form

$$A \leftarrow L_1, L_2, \ldots, L_k, c_1, c_2, \ldots, c_l,$$

where A is an atom, the L_i are literals and the c_i are constraints defined over some domain $\mathbb{D}$ of constraints. For our purposes, a *semiring-based constraint logic* *program (SCLP)* P consists of a finite set of clauses each of which is of the form

$$A \leftarrow L_1, L_2, \dots, L_k, \tag{1}$$

where A is an atom and the L_i are literals or is of the form

$$A \leftarrow a, \tag{2}$$

where A is an atom and a is any semiring value. Those clauses with a semiring value in the body constitute the constraints and are also known as "facts". The distinguished values $\bar{c}$ and $\underline{c}$ in a c-semiring are $\mathbf{1}$ and $\mathbf{0}$ respectively. Thus, when constructing P^* for a SCLP, unit clauses $A \leftarrow$ are replaced by $A \leftarrow \mathbf{1}$ and for any atom A not the head of a clause, we add the clause $A \leftarrow \mathbf{0}$ to P^*.

Example 3.1 Working over the natural numbers, we take $\mathbb{D} = (\mathbb{N}, con)$, where con is some set of constraints such as $x \le 5$, for example. Then the CLP clause $p(s(x)) \leftarrow p(x), x \le 5$ can be written as $p(s(x)) \leftarrow p(x), r(x)$ together with the clauses $r(o) \leftarrow$, $r(s(o)) \leftarrow$, $r(s^2(o)) \leftarrow$, $\dots$, $r(s^5(o)) \leftarrow$ all of which are taken to be in P. We then see that the following clauses are in P^*:

$$\begin{aligned}
p(s(o)) &\leftarrow p(o), r(o) \\
p(s^2(o)) &\leftarrow p(s(o)), r(s(o)) \\
&\vdots \\
p(s^6(o)) &\leftarrow p(s^5(o)), r(s^5(o)) \\
r(o) &\leftarrow \mathbf{1} \\
r(s(o)) &\leftarrow \mathbf{1} \\
r(s^2(o)) &\leftarrow \mathbf{1} \\
&\vdots \\
r(s^5(o)) &\leftarrow \mathbf{1} \\
r(s^6(o)) &\leftarrow \mathbf{0} \\
r(s^7(o)) &\leftarrow \mathbf{0} \\
&\vdots
\end{aligned}$$

In this context, a pre-interpretation $\mathcal{J}$ for a language $\mathcal{L}$ consists of a domain $\mathbb{D}$ together with the assignment of a mapping $f_{\mathcal{J}} : \mathbb{D}^n \to \mathbb{D}$ to each function symbol f in $\mathcal{L}$, where n is the arity of f. Furthermore, an interpretation I is a mapping $I : B_P \to \mathcal{S}$, and we denote by $I_{P,\mathcal{S}}$ the set of all such interpretations. Finally, associated with each SCLP is a consequence operator $T_{P,\mathcal{S}} : I_{P,\mathcal{S}} \to I_{P,\mathcal{S}}$ defined in [1] essentially as follows.

Definition 3.2 Given an interpretation I and a ground atom A, we define $T_{P,\mathcal{S}}(I)$ by

$$T_{P,\mathcal{S}}(I)(A) = \textstyle\sum_i I(C_i),$$

where $A \leftarrow \sum_i C_i$ is the unique pseudo-clause whose head is A, and $I(C_i)$ is defined as follows. We set $I(C_i) = a$ when $A \leftarrow \sum_i C_i$ is the fact $A \leftarrow a$, and otherwise when $A \leftarrow \sum_i C_i$ is not a fact of the form $A \leftarrow a$, we set $I(C_i) = \prod_{j=1}^{n_i} I(L^i_j)$, where $C_i = L^i_1, \ldots, L^i_{n_i}$, say.

It is easy to see that if P is a SCLP, then the general semantic operator $\mathfrak{T}_{P,\mathfrak{C}}$ coincides with $T_{P,\mathcal{S}}$ when we take $\mathfrak{C}$ to be the c-semiring $\mathcal{S}$ underlying P.

We close this discussion of constraint problems in relation to the general semantic operator by considering a simple example of a constraint problem that can be turned into a SCLP (and can ultimately be solved on an ANN, see Theorem 5.1). This example shows that we can solve a CSP, combine constraints and project over a subset of variables using methods from logic programming.

Example 3.3 In this example, the required CSP is defined over a constraint system $CS = (\mathcal{S}, \mathbb{D}, \mathcal{V})$, where $\mathbb{D} = \{a, b, c\}$, $\mathcal{V} = \{x, y, z\}$ and $\mathcal{S}$ is the classical-logic semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$. The problem (C, var) we address consists of four constraints: $C_1 =< def_1, \{x\} >$, $C_2 =< def_2, \{x, y\} >$, $C_3 =< def_3, \{y\} >$ and $C_4 =< def_4, \{y, z\} >$ and the set of variables $var = \{x, y, z\}$. The functions def_i are defined in the table below. Here, the logic is restricted to classical two-valued logic; thus, the constraints are either satisfied (indicated by value 1) or they are not (indicated by value 0).

Table 2
Constraints

def_1	def_2	def_3	def_4
$< a >= 1$	$< a, a >= 0$	$< a >= 0$	$< a, a >= 0$
$< b >= 1$	$< a, b >= 1$	$< b >= 1$	$< a, b >= 1$
$< c >= 0$	$< a, c >= 1$	$< c >= 1$	$< a, c >= 1$
	$< b, a >= 0$		$< b, a >= 0$
	$< b, b >= 0$		$< b, b >= 1$
	$< b, c >= 1$		$< b, c >= 1$
	$< c, a >= 0$		$< c, a >= 0$
	$< c, b >= 1$		$< c, b >= 1$
	$< c, c >= 0$		$< c, c >= 1$

A solution for this CSP is an assignment to the three variables in var which satisfies all the constraints. We introduce the relation P_{xyz} with arity 3 to calculate the c-semiring value of each of the assignments to x, y and z; it is included in the SCLP as follows.

$$\begin{aligned}
C_1(a) &\leftarrow 1\\
C_1(b) &\leftarrow 1\\
C_1(c) &\leftarrow 0\\
C_2(a,a) &\leftarrow 0\\
C_2(a,b) &\leftarrow 1\\
&\vdots\\
C_4(c,c) &\leftarrow 1\\
P_{xyz}(a,a,a) &\leftarrow C_1(a)C_2(a,a)C_3(a)C_4(a,a)\\
P_{xyz}(a,a,b) &\leftarrow C_1(a)C_2(a,a)C_3(a)C_4(a,b)\\
P_{xyz}(a,b,a) &\leftarrow C_1(a)C_2(a,b)C_3(b)C_4(b,a)\\
&\vdots\\
P_{xyz}(c,c,c) &\leftarrow C_1(c)C_2(c,c)C_3(c)C_4(c,c)
\end{aligned}$$

3.3 A Simple Model of Uncertainty in Logic Programs

In [12], Stamate introduced a simple framework with which to model uncertainty in rule-based systems. We briefly consider this here.

Let $m \in \mathbb{N}$. Following [12], we define the logic L_m to be the set $\{\pm(\frac{n}{m}) \mid n = 0, \ldots, m\}$ of truth values together with disjunction $\vee$ taken as *max* relative to the usual ordering on the rational numbers, and conjunction $\wedge$ taken as *min* relative to the same ordering. Note that disjunction and conjunction are both associative, commutative and idempotent, and hence both are finitely determined by Theorem 2.3; indeed, L_m is a c-semiring in which $\mathbf{0} = -1$ and $\mathbf{1} = 1$. Furthermore, there are three distinguished truth values: 1 denotes true, -1 denotes false, and 0 denotes unknown. In this case, $\leq_+$ or $\leq_\vee$ is the truth ordering $\leq_t$ of [12], namely, the ordering induced on L_m by the usual linear ordering on the rational numbers, and the knowledge order $\leq_k$ is defined by

$$0 \leq_k -\frac{1}{m} \leq_k -\frac{2}{m} \leq_k \ldots -\frac{m-1}{m} \leq_k -1$$

and

$$0 \leq_k \frac{1}{m} \leq_k \frac{2}{m} \leq_k \ldots \frac{m-1}{m} \leq_k 1$$

with no other inequalities being present.

The sets L_m are then complete distributive lattices, and therefore the partial orders $\leq_\vee$ and $\leq_\wedge$ determined by the connectives are dual to each other.

Example 3.4 $L_4 = (\{-1, -\frac{3}{4}, -\frac{2}{4}, -\frac{1}{4}, 0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\}, max, min)$, and the knowledge order is the partial order shown in Figure 1.

Elements of L_m are allowed as literals in the bodies of the clauses in the programs discussed in [12], and therefore these programs are special cases of $\mathfrak{C}$-normal logic programs. Furthermore, valuations or interpretations I are defined as usual as

$$\begin{tikzcd}
-1 & & & & & & & & 1 \\
& -\frac{3}{4} \arrow[ul] & & & & & & \frac{3}{4} \arrow[ur] & \\
& & -\frac{2}{4} \arrow[ul] & & & & \frac{2}{4} \arrow[ur] & & \\
& & & -\frac{1}{4} \arrow[ul] & & \frac{1}{4} \arrow[ur] & & & \\
& & & & 0 \arrow[ul] \arrow[ur] & & & &
\end{tikzcd}$$

Fig. 1. Partial Order $\leq_k$ on L_4.

functions from $B_P \cup L_m \to L_m$, where $I(l) = l$ for each $l \in L_m$, and a semantic operator Φ_P^S is defined in [12] by

$$\Phi_P^S(I)(A) = sup_{\leq_t}\{I(C) \mid A \leftarrow C \in \text{ground}(P)\}.$$

As usual, we take $lub_{\leq_t}(\emptyset)$ to be the least element -1 for the order $\leq_t$, and then the operator Φ_P^S is well-defined. Hence, -1 is taken to be the default value in this setting. Since each L_m is a complete lattice, the least upper bound of any set of truth values exists and is equal to their infinite disjunction. Indeed, such an infinite disjunction is equal to a finite disjunction by Theorem 2.3, and it now follows easily that Φ_P^S coincides with $\mathfrak{T}_{P,\mathfrak{C}}$ when $\mathfrak{C}$ is taken to be L_m.

Example 3.5 Stamate [12] only considers databases, that is, programs with no function symbols of arity greater than 0, and hence ground(P) and P^{**} are both finite. Let P be the following database, with three constants and two relation symbols.

$$\begin{aligned}
&p(a) \leftarrow \neg p(b)\\
&p(b) \leftarrow q(a)\\
&q(a) \leftarrow \neg p(a), p(c)\\
&q(b) \leftarrow -\frac{1}{4}\\
&p(a) \leftarrow q(b), q(c)\\
&p(b) \leftarrow \neg q(c)
\end{aligned}$$

Then P^{**} is the following.

$$\begin{aligned}
&p(a) \leftarrow \neg p(b) \vee (q(b) \wedge q(c))\\
&p(b) \leftarrow q(a) \vee \neg q(c)\\
&p(c) \leftarrow -1\\
&q(a) \leftarrow \neg p(a) \wedge p(c)\\
&q(b) \leftarrow -\frac{1}{4}\\
&q(c) \leftarrow -1
\end{aligned}$$

4 Fixed-Point Properties of $\mathfrak{T}_{P,\mathfrak{C}}$

In this section, we briefly consider the monotonicity and continuity properties of the operator $\mathfrak{T}_{P,\mathfrak{C}}$. Thus, we suppose again throughout this section that the set $\mathfrak{C}$ is given, as in Section 2.1, and that P denotes an arbitrary normal logic program defined over $\mathfrak{C}$. For Theorem 4.6 to hold, it is necessary for the underlying set $\mathfrak{C}$ to be a complete partial order; thus, a least element with respect to $\leq_+$ must be present in $\mathfrak{C}$ (we add this element to $\mathfrak{C}$ if necessary). To calculate the least fixed point of $\mathfrak{T}_{P,\mathfrak{C}}$, it is common practice to iterate on the least element. However, if we require the least fixed point to coincide with any useful semantics, it will usually be necessary to choose the default value $\underline{c} \in \mathfrak{C}$ to be the least element in the ordering $\leq_+$.

Proposition 4.1 *If P is a definite program, then the operator $\mathfrak{T}_{P,\mathfrak{C}}$ defined over a lattice $\mathfrak{C}$ is monotonic.*

The preceding result is true for normal programs if the negation operator is itself monotonic, that is, if we have $\neg a \leq \neg b$ whenever $a \leq b$.

Corollary 4.2 *For any $\mathfrak{C}$-normal logic program P, the operator $\mathfrak{T}_{P,\mathfrak{C}}$ defined over a lattice with monotonic negation is itself monotonic.*

An example of a logic with monotonic negation is $\mathcal{FOUR}_k$, where $\mathcal{FOUR}_k$ is Belnap's four-valued logic with the knowledge ordering $\leq_k$. (It is then the case that $+$ coincides with the gullibility operator $\bigoplus$ and $\times$ coincides with the consensus operator $\bigotimes$ discussed by Fitting in [4].)

Proposition 4.3 *Negation defined on $\mathcal{FOUR}_k$ is monotonic with respect to $\leq_{\bigoplus}$.*

Proof. In $\mathcal{FOUR}_k$, $\neg b = b$, $\neg u = u$, $\neg t = f$, $\neg f = t$. Since b is the top element in the order $\leq_{\bigoplus}$, we have $s \leq_{\bigoplus} b$ for $s = t, u, b, f$ and hence $\neg s \leq_{\bigoplus} \neg b$ since $\neg b$ $(= b)$ is also the top element in $\leq_{\bigoplus}$. Likewise for u, u is the bottom element in the order $\leq_{\bigoplus}$ so that $u \leq_{\bigoplus} s$ for $s = t, u, b, f$, and so $\neg u \leq_{\bigoplus} \neg s$ also since $\neg u = u$ is the bottom element. Therefore, it remains to check inequalities involving t and f, but t and f are incomparable and hence there are no inequalities involving only those truth values. □

Proposition 4.4 *If P is definite, then $\mathfrak{T}_{P,\mathfrak{C}}$ defined over a c-semiring is monotonic.*

Corollary 4.5 *Suppose P is a definite program and that addition in $\mathfrak{C}$ distributes over multiplication (that is, $a \times (b + c) = (a \times b) + (a \times c)$ for all $a, b, c \in \mathfrak{C}$). Then $\mathfrak{T}_{P,\mathfrak{C}}$ is monotonic.*

The previous result holds because: if $c_1 \leq c_2$ and $d_1 \leq d_2$, then $c_1 \times d_1 \leq c_2 \times d_2$ and $c_1 + d_1 \leq c_2 + d_2$.

Theorem 4.6 *Suppose P is a definite program and that the underlying set $\mathfrak{C}$ is a complete partial order. Then whenever $\mathfrak{T}_{P,\mathfrak{C}} : I_{P,\mathfrak{C}} \to I_{P,\mathfrak{C}}$ is monotonic with respect to $\sqsubseteq_+$, it is Scott continuous.*

Proof. By monotonicity of $\mathfrak{T}_{P,\mathfrak{C}}$, we immediately have that $\mathfrak{T}_{P,\mathfrak{C}}(D)$ is a directed set and that $\sup(\mathfrak{T}_{P,\mathfrak{C}}(D)) \sqsubseteq_+ \mathfrak{T}_{P,\mathfrak{C}}(\sup D)$ for each directed set $D \subseteq I_{P,\mathfrak{C}}$. Thus, it remains only to show that $\mathfrak{T}_{P,\mathfrak{C}}(\sup D) \sqsubseteq_+ \sup(\mathfrak{T}_{P,\mathfrak{C}}(D))$ for each directed set $D \subseteq I_{P,\mathfrak{C}}$.

To establish this, let $\mathfrak{T}_{P,\mathfrak{C}}(\sup D)(A) = t_\beta$ for an arbitrary $A \in B_P$. Then $\sup D(\sum_j C_j) = t_\beta$, that is, $\sup_{I \in D} I(\sum_j C_j) = t_\beta$, where $A \leftarrow \sum_j C_j$ is the unique pseudo-clause in P^{**} with head A. Let $\sup(\mathfrak{T}_{P,\mathfrak{C}}(D))(A) = t_\alpha$ which is to say that $\sup_{I \in D}(\mathfrak{T}_{P,\mathfrak{C}}(I))(A) = t_\alpha$. Then $\mathfrak{T}_{P,\mathfrak{C}}(I)(A) \leq_+ t_\alpha$ for all $I \in D$, and from the definition of $\mathfrak{T}_{P,\mathfrak{C}}$, we have $I(\sum_j C_j) \leq_+ t_\alpha$ for all $I \in D$. Thus, t_α is an upper bound for $I(\sum_j C_j)$. But since t_β is the least upper, we must have $t_\beta \leq_+ t_\alpha$. Therefore, for any $A \in B_P$, $\mathfrak{T}_{P,\mathfrak{C}}(\sup D)(A) \sqsubseteq_+ \sup(\mathfrak{T}_{P,\mathfrak{C}}(D))(A)$, and it follows that $\mathfrak{T}_{P,\mathfrak{C}}(\sup D) \sqsubseteq_+ \sup(\mathfrak{T}_{P,\mathfrak{C}}(D))$, as required. Therefore, $\mathfrak{T}_{P,\mathfrak{C}}$ is Scott continuous. □

5 Connections with Artificial Neural Networks *ANN*

The following result was established in [9]. As indicated in the introduction, it connects neural networks with logic programming, constraint logic programming, and uncertainty by computing $\mathfrak{T}_{P,\mathfrak{C}}$ in each of these three cases (with the choices for $\mathfrak{C}$ made as in Section 3).

Theorem 5.1 *Suppose that both $+$ and $\times$ are finitely determined in $\mathfrak{C}$, and that P is a propositional logic program defined over $\mathfrak{C}$. Then there is a 3-layer feedforward neural network containing conjunction units in the second layer and disjunction units in the third layer (where conjunction and disjunction units are as as defined in [9]) which computes $\mathfrak{T}_{P,\mathfrak{C}}$.*

6 Conclusions

Inspired by a number of logic-based systems, we have shown how one may define a "semantic" operator in a purely algebraic way. Furthermore, we have also shown how it in fact encapsulates the logical semantics of a number of actual logic-based models of computation of current interest in the literature. Finally, we have addressed the monotonicity properties and the Scott continuity of this operator, and also, briefly, its computation by artificial neural networks.

References

[1] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(1):1–29, Jan 2001.

[2] Stefan Bonnier, Ulf Nilsson, and Torbjörn Näslund. A simple fixed point characterization of three-valued stable model semantics. *Information Processing Letters*, 40(2):73–78, 1991.

[3] Phan Minh Dung and Kanchana Kanchanasut. A fixpoint approach to declarative semantics of logic programs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Logic Programming, Proceedings of the North American Conference 1989, NACLP'89, Cleveland, Ohio*, pages 604–625. MIT Press, 1989.

[4] Melvin Fitting. Kleene's logic, generalized. *Journal of Logic and Computation*, 1:797–810, 1992.

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

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

[7] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *POPL'87: Proceedings of the Fourteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[8] John W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 1987.

[9] Máire Lane and Anthony K. Seda. Some aspects of the integration of connectionist and logic-based systems. *Information*, 9(4):551–562, 2006.

[10] Anthony K. Seda and Máire Lane. On approximation in the integration of connectionist and logic-based systems. In L. Li and K.K. Yen, editors, *Proceedings of The Third International Conference on Information, Information'04, Tokyo, November, 2004*, pages 297–300. International Information Institute, 2004.

[11] Anthony K. Seda and Máire Lane. On the measurability of the semantic operators determined by logic programs. *Information*, 8(1):33–52, 2005.

[12] Daniel Stamate. Quantitative datalog semantics for databases with uncertain information. In Alessandra Di Pierro and Herbert Wiklicky, editors, *Proceedings of the 4th Workshop on Quantitative Aspects of Programming Languages (QAPL 2006)*, Electronic Notes in Theoretical Computer Science, Vienna, Austria, April 1–2 2006. Elsevier. To appear.

[13] Matthias Wendt. Unfolding the well-founded semantics. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 53(12/s):56–59, 2002.