5-1990

Monotone Logic Programming

Howard A. Blair
Syracuse University, School of Computer and Information Science, blair@top.cis.syr.edu

Allen Brown Jr.

V. S. Subrahmanian

SU-CIS-90-08

Monotone Logic Programming

H. Blair, A. L. Brown, Jr., and V.S. Subrahmanian

May 1990

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

Monotone Logic Programming*

Howard A. Blair†
Allen L. Brown, Jr.‡
V. S. Subrahmanian§

Abstract

We propose a notion of an *abstract logic*. Based on this notion, we define abstract logic programs to be sets of sentences of an abstract logic. When these abstract logics possess certain logical properties (some properties considered are compactness, finitariness, and monotone consequence relations) we show how to develop a fixed-point, model-state-theoretic and proof theoretic semantics for such programs. The work of Melvin Fitting on developing a generalized semantics for multivalued logic programming is extended here to arbitrary abstract logics. We present examples to show how our semantics is robust enough to be applicable to various non-classical logics like temporal logic and multivalued logics, as well as to extensions of classical logic programming such as disjunctive logic programming. We also show how some aspects of the declarative semantics of distributed logic programming, particularly work of Ramanujam, can be incorporated into our framework.

1 Introduction

When logic programming was initially introduced by Kowalski, programs were restricted to be finite sets of definite clauses. The rationale behind this restriction was that a reasonably simple and fast proof procedure existed for this fragment of classical logic.

While it has been proved that definite clause programs are computationally universal (cf. Andreka and Nemeti [2] and Blair [7, 8]), definite clauses lack expressive power. Thus, various researchers were led to investigate the semantics of various syntactic extensions of definite clause programs. For example, clause bodies may be permitted to contain negated atoms, clause heads may be allowed to contain disjunctions of atomic formulas, and so on.

A different kind of extension to classical definite clause logic programming occurs when we allow logic programs to be sets of formulas in some non-classical logic. Such a formalism

*Address correspondence to V.S. Subrahmanian.

†School of Computer and Information Science, 4–116 Center for Science and Technology, Syracuse University, Syracuse, NY 13244-1240, U.S.A. Electronic Mail: blair@top.cis.syr.edu

‡Systems Sciences Laboratory, Xerox Webster Research Center, 800 Phillips Road, Webster, NY 14580, U.S.A. Electronic Mail: abrown.wbst@xerox.com

§Department of Computer Science, A. V. Williams Building, University of Maryland, College Park, MD 20742, U.S.A. Electronic Mail: vs@mimsy.umd.edu

has been found to be useful for incorporating quantitative deduction into logic programming. Similarly, various non-classical logic programming languages have been introduced for reasoning in the presence of temporal and modal phenomena, for reasoning with inconsistency, reasoning about beliefs, etc.

In those cases where the semantics of such extensions has been studied, it has been found that these different extensions are semantically similar.

In this paper, we will consider an abstract logic to be an appropriately defined mathematical structure. Programs will then turn out to be sets of objects in this structure. We will then demonstrate how to associate with each program, an operator that maps sets of formulas (which will be defined relative to the structure) to sets of formulas. Various relations between the fixed-points of this operator, and properties of the corresponding structure will be established. A generalized resolution scheme is developed for these abstract logics, and various soundness and /or completeness results are proved.

We will show how various logic programming formalisms existing in the current literature can be incorporated within our theoretical framework. Examples include: classical logic programming, temporal logic programming, bilattice based logic programming, paraconsistent logic programming and quantitative logic programming.

2 Abstract Logics

Following a standard convention, given a set Σ of symbols, we use the notation Σ^+ to denote the set of non-empty strings generated by this alphabet.

An *abstract logic* $\mathcal{L}$ consists of:

1. a non-empty set (possibly infinite) $\mathcal{L}_\Sigma$ of symbols
2. an infinite set $\mathcal{L}_{mv}$ of symbols called *meta-variables* such that $(\mathcal{L}_\Sigma \cap \mathcal{L}_{mv}) = \emptyset$ and
3. a set $\mathcal{L}_{\text{wff}} \subseteq \mathcal{L}_\Sigma^+$ of objects called *well-formed formulas* (or wffs, for short)
4. a set $\mathcal{L}_{mt} \subseteq (\mathcal{L}_\Sigma \cup \mathcal{L}_{mv})^+$ of objects called *meta-terms* satisfying the following two conditions:

   (a)

   $$(\mathcal{L}_{wff} \cup \mathcal{L}_{mv}) \subseteq \mathcal{L}_{mt}$$

   and

   (b) if V is a meta-variable that occurs in some meta-term $T \in \mathcal{L}_{mt}$, then the string obtained by replacing V in T by formula F is also a meta-term.

5. a set $\mathcal{L}_{ir}$ of syntactic entities, called *inference rules*, of the form

   $$\frac{Numerator}{T_0}$$

where $Numerator$ is a subset of $\mathcal{L}_{mt}$ and T_0 is a meta-term. The cardinalioty of $Numerator$ is called the *order* of the inference rule, and T_0 is called the *denominator* of this inference rule.

Thus, a logic is a 5-tuple $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$. $\mathcal{L}$ is said to be *finitary* iff each rule in $\mathcal{L}_{ir}$ has a numerator of finite order. Notice that there is no need for well-formed formulas of logic $\mathcal{L}$ to look like the familiar syntactic well-formed formulas of classical logic. $\mathcal{L}_{wff}$ is an arbitrary set which we may choose in any way we please. We may *choose* $\mathcal{L}_{wff}$ to be, say:

1. the set of wffs of classical logic or
2. the set of wffs of modal logic or
3. the set of wffs of auto-epistemic logic or
4. the definite clause fragment of classical logic, etc.

From here on, we assume that $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is some arbitrary, but fixed, abstract logic.

Definition 2.1 A *meta-variable substitution* is a function from $\mathcal{L}_{mv}$ to $\mathcal{L}_{wff}$. If σ is a meta-variable substitution, then we extend σ to a function σ' of type $(\mathcal{L}_{mt} \longrightarrow \mathcal{L}_{wff})$ as follows:

1. If $T \in \mathcal{L}_{wff}$, then $\sigma'(T) = T$
2. If $T \in \mathcal{L}_{mv}$, then $\sigma'(T) = \sigma(T)$.
3. Otherwise, $\sigma'(T)$ is the well-formed formula obtained by replacing each meta-variable V occurring in T by $\sigma(V)$. (Note: Condition 2 in the definition of meta-terms ensures that $\sigma(T)$ is a meta-term, and also, of course, that aplication of a meta-variable substitution to a meta-term eliminates all meta-variables.)

Given a metavariable-substitution σ, the extension σ' of σ is unique, and therefore, we will often abuse terminology and assume σ can be applied to meta-terms. Thus, if T is a meta-term, then we will use the notation $\sigma(T)$ to denote $\sigma'(T)$.

Definition 2.2 If $\frac{Numerator}{T}$ is an inference rule of $\mathcal{L}$, and σ is a metavariable-substitution, then $\frac{\{\sigma(F) \mid F \in Numerator\}}{\sigma(T)}$ is said to be an *instance* of the above inference rule of $\mathcal{L}$.

If X is a set, we use the notation $\mathcal{P}(X)$ to denote the power set of X.

Definition 2.3 Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic. For a set $X \subseteq \mathcal{L}_{\text{wff}}$ of wffs, a *derivation* from X is a finite or transfinite sequence

$$D = \langle F_0, F_1, \ldots, F_\alpha, \ldots \rangle$$

such that for each $F_\gamma \in D$, either $F_\gamma \in D$ or there is an inference rule in $\mathcal{L}_{ir}$ having an instance of the form $\frac{Numerator}{F_\gamma}$ where $Numerator \subseteq \{F_\beta \mid 0 \leq \beta < \gamma\}$.

The binary relation $\vdash_{\mathcal{L}}$ between the power set $\mathcal{P}(\mathcal{L}_{\text{wff}})$ of $\mathcal{L}_{\text{wff}}$ and $\mathcal{L}_{\text{wff}}$ is defined by: $X \vdash F$ iff F occurs in a derivation from X.

When $\mathcal{L}$ is clear from context, we will abuse notation and simply write $\vdash$ instead of $\vdash_{\mathcal{L}}$.

We will often abuse notation as follows: Given $X, Y \subseteq \mathcal{L}_{wff}$, we may write $X \vdash Y$ to denote that $X \vdash \phi$ for all $\phi \in Y$.

One may wish to think of $\vdash$ as a syntactic entailment relation. Note however that there is no restriction limiting proofs to be finitary (or even effective). For logics in which a property similar to the compactness theorem of classical logic holds, infinitary proofs are not necessary. Our aim in allowing infinitary proofs is simply to increase the level of generality of our framework.

Before we proceed to discuss logic programs written in abstract logics, we present a few examples of abstract logics.

Example 2.1 Classical logic under the usual rules of inference (cf. Schoenfield [34]) is an abstract logic.

Example 2.2 Let $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ be the logic whose well formed formulas are definite clauses over some pre-determined alphabet of *non-logical* symbols (cf. Shoenfield [34]), whose set $\mathcal{L}_{mv}$ of meta-variables is $\{\Gamma_1, \Gamma_2, \ldots\}$, whose set of meta-terms is $(\mathcal{L}_{mv} \cup \mathcal{L}_{wff})$ and which has the single inference rule:

$$\frac{\Gamma_1 \ , \ \Gamma_2 \leftarrow \Gamma_1}{\Gamma_2}$$

Then $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic.

Remark 2.1 Let $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ be any logic. Then: $\vdash$ is monotone, i.e. if $X_1 \subseteq X_2 \subseteq \mathcal{L}_{wff}$ and $X_1 \vdash F$, then $X_2 \vdash F$. □

In general, $\vdash$ may not be continuous, i.e. there may be a sequence $X_1 \subseteq X_2 \subseteq \cdots$ of subsets of $\mathcal{L}_{wff}$ and a formula F such that $(\bigcup_i X_i) \vdash F$ but there is no j such that $X_j \vdash F$. To see this, let $\mathcal{L}$ be any logic that does not possess the well known compactness property [34].

3 Abstract Logic Programs

Definition 3.1 An *abstract logic program* (in logic $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$) is a finite subset of $\mathcal{L}_{wff}$.

Definition 3.2 A *theory* (in logic $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$) is any subset TH of $\mathcal{L}_{wff}$ satisfying the condition that if $X \subseteq TH$ and $X \vdash F$, then $F \in TH$.

In other words, we view an abstract logic program as the finite set of non-logical axioms of a theory in an abstract logic.

All the definitions described thus far are fairly general. No restrictions imposed thus far prevent us from considering logic programs to be arbitrary finite sets of first order formulas. Or for that matter, nothing said so far prevents us from using full first order modal logic (say first order $S5$) as a logic programming language. We will see below why we need to impose the restriction that abstract logic programs are finite.

Definition 3.3 Suppose P is a set of wffs (w.r.t. logic $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$). We associate an operator, denoted T_P, with P. T_P maps the power set, $\mathcal{P}(\mathcal{L}_{wff})$ of $\mathcal{L}_{wff}$ to $\mathcal{P}(\mathcal{L}_{wff})$ and is defined as follows: $F \in T_P(X)$ iff either $F \in P$ or $\frac{Numerator}{F}$ is an instance of an inference rule of $\mathcal{L}$ and $Numerator \subseteq X$.

Suppose we consider the case of classical logic programming without negation ([25]). In this case, the logic $\mathcal{L}$ is classical logic with the usual rules of inference, and P is always a finite set of definite clauses. Suppose $X = \emptyset$. Then $T_P(X)$ turns out to be $P \cup TAUT$ where $TAUT$ is the set of tautologies of classical logic. At this point, the only ground atoms in T_P are those unit ground clauses in P; note that as $X = \emptyset$, the rule of instantiation was not applied. However, $T_P(T_P(\emptyset))$ gives us all the ground atoms obtained after the first application of the classical T_P operator (Lloyd [25]). In effect, our T_P operator simulates the classical operator in two steps – the first step corresponding to modus ponens, and the second to instantiation.

Note that T_P is defined when P is any set of wffs in the abstract logic $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$. However, usually, we will assume that P is an abstract logic program, i.e. P is finite.

Proposition 3.1 Suppose $P \subseteq \mathcal{L}_{\text{wff}}$ (in logic $\mathcal{L}$). Then: T_P is monotone. □

Given an abstract logic $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$, we will take the set $\mathcal{P}(\mathcal{L}_{wff})$ to be a complete lattice when ordered by set inclusion.

Proposition 3.2 Suppose P is any logic program in a *finitary* abstract logic $\mathcal{L}$. Then T_P is continuous and hence $\vdash$ is continuous. □

Note that in general, the above theorem does not hold for logic programs over arbitrary logics. Finitariness is required. Blair and Subrahmanian [11] show that their T_P operator

for annotated logic programs is not always continuous. The annotated logic which serves as such a counterexample uses an infinite valued complete lattice $\mathcal{T}$ which they construct, together with, for each subset $\mathcal{T}' \subseteq \mathcal{T}$, the inference rule

$$\frac{\{\mathbf{A} : \mu \mid \mu \in \mathcal{T}'\}}{\bigsqcup_{\rho \in \mathcal{T}'} \rho}.$$

As $\mathcal{T}'$ may be infinite, it is easy to see that such a logic is not finitary.

Definition 3.4 Let $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ be an abstract logic and P an abstract logic program. A set X, $P \subseteq X \subseteq \mathcal{L}_{wff}$ is said to be a *P-model-state* of $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ iff for every instance $\frac{Numerator}{G}$ of an inference rule $R \in \mathcal{L}_{ir}$ such that $Numerator \subseteq X$ $G \in X$.

Proposition 3.3 Suppose P is an abstract logic program. X is a P-model-state of the abstract logic $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ iff $T_P(X) \subseteq X$. □

The above theorem establishes a one to one correspondence between P-model-states and the pre-fixed-points of the T_P operator.

Definition 3.5 Suppose $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic, P is a logic program (w.r.t. $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$), and $X \subseteq \mathcal{L}_{wff}$. X is said to be *P-supported* iff whenever $F \in X$, then either $F \in P$ or there is an inference rule $R \in \mathcal{L}_{ir}$ having an instance of the form $\frac{Numerator}{F}$ such that $Numerator \subseteq X$.

Proposition 3.4 Suppose $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic, and P is an abstract logic program and $X \subseteq \mathcal{L}_{wff}$. X is P-supported iff $T_P(X) \supseteq X$.

Proof. X is P-supported

iff $(\forall F \in \mathcal{L}_{wff})$ if $F \in X$ then either $F \in P$ or there is an inference rule in $\mathcal{L}$ having an instance of the form $\frac{Numerator}{F}$ such that $Numerator \subseteq X$

iff $(\forall F \in \mathcal{L}_{wff})$ if $F \in X$ then $F \in T_P(X)$

iff $X \subseteq T_P(X)$. □

Theorem 3.1 Suppose $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic, and P is an abstract logic program and $X \subseteq \mathcal{L}_{wff}$. X is a P-supported model-state of $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ iff $T_P(X) = X$. □

Theorem 3.1 is an immediate consequence of Proposition 3.3 and 3.4. It establishes a one-one correspondence between the fixed-points of the T_P operator and the supported model-states of the program P. Given a logic $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$, and an abstract logic program P in $\mathcal{L}$, we now define the iterations of the T_P operator.

Definition 3.6 (Transfinite Iteration of T_P operator) Let λ be an ordinal. Define:

$$T_P \uparrow 0 = \emptyset \qquad T_P \downarrow 0 = \mathcal{L}_{wff}$$
$$T_P \uparrow \lambda = \bigcup_{\gamma<\lambda} T_P(T_P \uparrow \gamma) \qquad T_P \downarrow \lambda = \bigcap_{\gamma<\lambda} T_P(T_P \downarrow \gamma)$$

Proposition 3.5 Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic. Furthermore, suppose P is a logic program (in logic $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$). Then: $P \vdash F$ iff $F \in lfp(T_P)$. ($lfp(T_P)$ denotes the least fixed-point of the operator T_P).

Proof. We proceed in two parts.

I. Suppose $P \vdash F$. Then there is a sequence $F_0, F_1, F_2, \ldots, F_\alpha$ where α is some ordinal such that $F_\alpha = F$ and this sequence satisfies the conditions of Definition 2.3. We proceed by transfinite induction on α. We show that $F \in lfp(T_P)$.

Base Case. ($\alpha = 0$) Trivial.

Inductive Case. Let $\Gamma = \{F_\gamma \mid \gamma < \alpha\}$. Then, by the induction hypothesis, $\Gamma \subseteq lfp(T_P)$. By Definition 2.3, there is an inference rule of $\mathcal{L}$ having an instance of the form

$$\frac{Numerator}{F_\alpha}$$

such $Numerator \subseteq \Gamma \subseteq lfp(T_P)$. Hence, by definition of T_P, $F \in T_P(lfp(T_P)) = lfp(T_P)$.

Thus we have shown that if $P \vdash F$, then $F \in lfp(T_P)$.

II. We now show the reverse implication, i.e if $F \in lfp(T_P)$, then $P \vdash F$. As T_P is monotone, and as $\mathcal{P}(\mathcal{L}_{wff})$ is a complete lattice under inclusion, it is easy to see that T_P has a least fixed point and that this fixed-point is identical to $T_P \uparrow \gamma$ for some ordinal γ. We proceed by induction on γ.

Base Case. ($\gamma = 0$) Trivial.

Inductive Case. Suppose $F \in T_P \uparrow \gamma$. There are two possibilities:

If γ is a limit ordinal, then $F \in \bigcup_{\psi<\gamma} T_P \uparrow \psi$ and therefore, there is some ordinal $\phi < \gamma$ such that $F \in T_P \uparrow \phi$. Therefore, by the induction hypothesis, $P \vdash F$.

If γ is a successor ordinal, i.e. $\gamma = (\delta + 1)$ for some ordinal δ, then there is an inference rule of $\mathcal{L}$ having an instance of the form

$$\frac{Numerator}{F}$$

where $Numerator = \{T_0, \ldots, T_\zeta, \ldots\}$ such that $P \vdash T_i$ for all $T_i \in Numerator$. Let Λ_i be the derivation sequence of T_i from P. Then

$$\Lambda_1 \cdots \Lambda_\zeta \cdots F$$

is a derivation (possibly infinite) of F from P. Hence, $P \vdash F$.

Thus we have shown that if $F \in lfp(T_P)$ then $P \vdash F$. This completes the proof. □

In general, T_P is not necessarily continuous, in particular because $\mathcal{L}$ need not be finitary, and therefore, α may be greater than ω.

Definition 3.7 Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic and let X be a subset of $\mathcal{L}_{wff}$. X is said to be *compact* iff whenever $X \vdash F$, it is the case that there is a finite subset X' of X such that $X' \vdash F$. $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is said to be *compact* iff each subset $X \subseteq \mathcal{L}_{wff}$ in $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is compact.

Theorem 3.2 Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is a *finitary* abstract logic and X is a subset of $\mathcal{L}_{wff}$. Then: $T_X \uparrow \omega = lfp(T_X) = \{F \mid X \vdash F\}$. Note here that X is an arbitrary subset of $\mathcal{L}_{wff}$, and hence may not be finite.

Proof. Immediate consequence of Propositions 3.2 and 3.5. □

This theorem makes us ask: "What kinds of logics are finitary ? Are most logics likely to interest us finitary ?" Some examples of finitary logics are:

1. classical first order logic
2. most systems of modal logic (e.g. S4,S5, and K).
3. most systems of first order temporal logic (e.g. those described in [5, 21])
4. dynamic logic (cf. Harel [23])
5. most systems of many-valued logics possessing a *finite* set of truth values

The fact that some of the above-mentioned logics have a highly undecidable validity problem is irrelevant to the finitariness of the logics. Note that finitariness only says that each inference rule of the logic $\mathcal{L}$ contains a numerator of finite order. However, the collection $\mathcal{L}_{ir}$ of inference rules may itself be highly undecidable or of very high cardinality. Moreover, the completeness theorems we will prove in Section 4 relate to the *existence* of proofs, and not to whether they can be effectively found.

It is also instructive to give an idea of some logics that are *not* finitary. One example is the system of infinite-valued annotated logics due to Subrahmanian [37]. These logics use an infinitary inference rule that may be summed up as follows: "If A is an atom and $(\mu_i)_{i \in \mathcal{A}}$ are truth values, then from the set of annotated atoms $(A : \mu_i)_{i \in \mathcal{A}}$, conclude $A : \mu$ where $\mu = \sqcup_{i \in \mathcal{A}} \mu_i$." Further details of this rule and its logical aspects may be found in [14]. Another well known logic that is not finitary is second order logic.

The results obtained thus far relate to the declarative semantics of logic programming with different logics. Below, we demonstrate how a few examples of extensions and variations of the logic programming paradigm (currently existing in the literature) fit into our framework for programming with abstract logics.

Example 3.1 (Classical Logic Programming) In order to model classical logic programming as an instance of our scheme, we need to consider only the following rules of inference:

$$\frac{\mathbf{A}\ ,\ \mathbf{B} \leftarrow \mathbf{A}}{\mathbf{B}}$$

$$\frac{(\forall x)F[x]}{F[t]}$$

where $F[x]$ denotes a formula F containing zero or more free occurrences of a variable symbol x, and $F[t]$ denotes the replacement of x in F by a term t.

Example 3.2 (Disjunctive Logic Programming) In this case, for each integer $n \geq 0$, we need the following rule of inference:

$$\frac{\mathbf{A}_1 \vee \mathbf{B}_1, \ldots, \mathbf{A}_n \vee \mathbf{B}_n,\ ,\ \mathbf{C} \leftarrow \mathbf{B}_1 \,\&\, \cdots \,\&\, \mathbf{B}_n}{\mathbf{A}_1 \vee \cdots \vee \mathbf{A}_n \vee \mathbf{C}}$$

With these rules of inference (this is basically hyper-resolution), together with the instantiation rule for classical logic, it is now possible to model the declarative semantics of disjunctive logic programming as described by Minker and Rajasekar [29].

Example 3.3 (Multivalued Logic Programming) Consider quantitative logic programs as defined by Blair and Subrahmanian [9, 10]. Here, we may use the following rules of inference:

$$\frac{}{\mathbf{A} : \bot}$$

where $\bot$ is the least element of the complete lattice being considered. In addition, for each subset $\mathcal{T}' \subseteq \mathcal{T}$, we need the rule:

$$\frac{\{\mathbf{A} : \mu \mid \mu \in \mathcal{T}'\}}{\mathbf{A} : \bigsqcup_{\mu \in \mathcal{T}'} \mu}.$$

The rule for modus ponens is also necessary:

$$\frac{\mathbf{A}\ ,\ \mathbf{B} \leftarrow \mathbf{A}}{\mathbf{B}}$$

Example 3.4 (Temporal Logic Programming) We consider the linear time temporal logic programming language defined by Baudinet [5]. She augments classical logic with two new operators, $\Box$ (necessity) and $\bigcirc$ (the "next time" operator). Intuitvely, $\Box p$ is true at time t iff p is true at all times $t' \geq t$. $\bigcirc p$ is true at time t iff p is true at time $(t+1)$.

If A is an atomic formula of classical logic, then $\underbrace{\bigcirc \ldots \bigcirc}\, k$ times$A$ where $k \geq 0$ is called a *next-atom.* An initial program clause is a sentence of the form

$$N_0 \leftarrow N_1 \,\&\, \ldots \,\&\, N_k$$

where each N_i, $0 \leq i \leq k$, is a next-atom. If C is an initial program clause, then $\Box C$ is a *permanent* program clause. A temporal logic program is a finite collection of initial and

permanent program clauses. It can easily be proved that any temporal logic program is logically equivalent to a (possibly infinite) set of initial program clauses.

In this setting, the inference rules to be used are:

$$\frac{\Box\phi}{\bigcirc^i\phi}$$

$$\frac{\phi\ ,\ \psi \leftarrow \phi}{\psi}$$

where the first inference rule is actually a scheme for an infinite set of inference rules obtained by instantiating i to various different integers. In addition, we need the rule:

$$\frac{\bigcirc(\phi\ \&\ \psi)}{\bigcirc\phi\ \&\ \bigcirc\psi}.$$

Baudinet's [5] proof procedure for temporal logic programming provides an explanation for why these proof systems are complete.

Example 3.5 (Distributed Logic Programming) If $P_1, \ldots, P_n$ are pure logic programs, then the n-tuple $P = (P_1, \ldots, P_n)$ is called a distributed logic program. Each of $\{1, \ldots, n\}$ is called a *site*. Program P_i is said to be "the" program at site i. Ramanujam [32] has developed an elegant model theoretic, fixed-point and proof theoretic semantics for distributed logic programs of this sort. In our abstract logic framework, wffs may be considered to be n-tuples $(F_1, \ldots, F_n)$ where each F_i is either a clause or a conjunction of ground atoms or a special dummy symbol $\sharp$. Thus, if we wish to say that p is true at site 2 and nothing else is stated about the truth of formulas at other sites, then we assert the wff

$$\underbrace{(\sharp, p, \sharp, \ldots, \sharp)}_{n\text{ tuple}}.$$

For the sake of convenience, suppose $n = 2$. To say that p and q are both true at site 2 and r is true at site 1, we have two options. We could assert the single tuple:

$$(r, p\ \&\ q)$$

or we could assert two tuples,

$$(r, p)\ ,\ (r, q).$$

The inference rules for our logic consist of the following:

$$(\mathbf{MP_0})\quad \frac{(\mathbf{A_1}, \ldots, \mathbf{A_{i-1}}, \mathbf{A_i}, \mathbf{A_{i+1}}, \ldots, \mathbf{A_n})\ ,\ (\mathbf{A_1}, \ldots, \mathbf{A_{i-1}}, (\mathbf{B} \leftarrow \mathbf{A_i}), \mathbf{A_{i+1}}, \ldots, \mathbf{A_n})}{(\mathbf{A_1}, \ldots, \mathbf{A_{i-1}}, \mathbf{B}, \mathbf{A_{i+1}}, \ldots, \mathbf{A_n})}$$

Such a rule is in $\mathcal{L}_{ir}$ for each i, $1 \leq i \leq n$. Intuitively, this rule says that if some site, say site j, knows that A is true, and $B \leftarrow A$ is a clause in site i, then site i knows that B is true. In particular, a site cannot know A if there is no clause in that site having A as a ground instance of its head. In addition, for each pair of integers $1 \leq i, j \leq n$, we have a rule of the form:

$$(\mathbf{MP_1})\quad \frac{(\mathbf{A_1}, \ldots, \mathbf{A_{i-1}}, \mathbf{A_i}, \mathbf{A_{i+1}}, \ldots, \mathbf{A_{j-1}}, \mathbf{B} \leftarrow \mathbf{A_i}, \mathbf{A_{j+1}}, \ldots, \mathbf{A_n})}{\mathbf{A_1}, \ldots, \mathbf{A_{i-1}}, \mathbf{A_i}, \mathbf{A_{i+1}}, \ldots, \mathbf{A_{j-1}}, \mathbf{B}, \mathbf{A_{j+1}}, \ldots, \mathbf{A_n})}$$

This rule captures the intuition that an atom A becomes true at site i iff there is a clause in P_i having a ground instance of the form

$$A \leftarrow B_1 \,\&\, \ldots \,\&\, B_k$$

such that each B_j is true in some site P_j, $1 \leq j \leq n$. (Note that in rule $(\mathbf{MP_1})$ $\mathbf{A_i}$ is assumed to be an atom. This assumption is made for the sake of simplicity; stating this rule when $\mathbf{A}_i$ is a conjunction of atoms is straightforward, though notationally cumbersome, because the different atoms in $\mathbf{A}_i$ are allowed to be true at different sites). In addition, for each $1 \leq i \leq n$, we have the rule

$$\frac{(\mathbf{A}_1, \ldots, \mathbf{A}_j, \ldots, \mathbf{A}_n) \ , \ (\mathbf{B}_1, \ldots, \mathbf{B}_j, \ldots, \mathbf{B}_n)}{(\mathbf{A}_1, \ldots, (\mathbf{A}_j \,\&\, \mathbf{B}_j), \ldots, \mathbf{B}_n)}$$

Repeated application of this rule allows us to derive the following inference rule

$$\frac{(\mathbf{A}_1, \ldots, \mathbf{A}_j, \ldots, \mathbf{A}_n) \ , \ (\mathbf{B}_1, \ldots, \mathbf{B}_j, \ldots, \mathbf{B}_n)}{(\mathbf{A}_1 \,\&\, \mathbf{B}_1, \ldots, \mathbf{A}_n \,\&\, \mathbf{B}_n)}$$

This formulation is sufficient to capture the declarative semantics for distributed logic programming proposed by Ramanujam [32]. The operator associated with programs written in this abstract logic allows is to capture the exact same fixed-point theory described by Ramanujam.

One of the crucial aspects of the distributed logic programming formalism of Ramanujam [32] that we do *not* capture is synchronization. Ramanujam constructs an elegant proof theoretic semantics in which synchronization operators are introduced. We do not know how to mimic this behavior that forms an essential ingredient of his semantics.

We believe that these examples demonstrate that our framework holds some promise of yielding, in the long run, a unifying treatment of classical and non-classical logic programming semantics.

4 "Proof Theoretic Semantics"

Definition 4.1 Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic, F is a formula, and $G, W \subseteq \mathcal{L}_{wff}$. We say formula H is a *generalized resolvent* of F and G iff there is an inference rule in $\mathcal{L}_{ir}$ having an instance of the form

$$\frac{Numerator}{H}$$

such that $F \in Numerator$ and $Numerator - \{F\} \subseteq G$. H is also called a *unary resolvent* of W if $Numerator \subseteq W$.

Given F and G in $\mathcal{L}_{wff}$, we denote the set of generalized resolvents of F and G by $\Re(F, G)$. Suppose now that $\leq$ is a reflexive ordering on $\mathcal{L}_{wff}$. We use the notation $\Re_{\leq}(F, G)$ to denote the minimal elements, if any, of $\Re(F, G)$. (Observe that minimal elements may not exist. Also observe that in general, we may not interchange the arguments of $\Re$.)

Example 4.1 Suppose we are considering (classical) clausal logic, i.e. $\mathcal{L}_{wff}$ is the set of all clauses (including the empty clause) generated by some alphabet. Then the generalized resolvent of two clauses is of the form $(R \vee D)$ where R is the resolvent of the two clauses we are considering and D is any disjunction (possibly empty). We may assume that $\mathcal{L}_{ir}$ contains the inference rule:

$$\frac{\mathbf{C_1} \vee \mathbf{A} \vee \mathbf{C_2} \ , \ \mathbf{D_1} \vee \neg\mathbf{A} \vee \mathbf{D_2}}{\mathbf{C_1} \vee \mathbf{D_1} \vee \mathbf{C_2} \vee \mathbf{D_2}}$$

where $\mathbf{C_1}, \mathbf{C_2}, \mathbf{D_1}, \mathbf{D_2}$ are all scheme variables. If we assume that clauses are ordered by the subsumption orderingi (i.e. $C_1 \leq C_2$ iff C_1 subsumes C_2), then the minimal clauses are just the ordinary resolvents (cf. Robinson [33]).

Consider the following two clauses $C_1 \equiv D_1 \vee A_1 \vee D_2$ and $C_2 \equiv E_1 \vee \neg A_2 \vee E_2$ where D_1, D_2, E_1 and E_2 are all disjunctions (possibly empty). Then in order to appply the above inference rule, we need to find an instance of C_1 and C_2 such that A_1 and A_2 yield the same literal (the inference rule above applies only in this case). But this is just the same as unification. In particular, the above inference rule allows arbitrary unifiers to be used in resolution, not just mgu's.

Definition 4.2 Suppose $X \subseteq \mathcal{P}(\mathcal{L}_{wff})$, and P is an abstract logic program. A sequence $F_1, F_2, \ldots, F_\alpha$ is called a *generalized X-linear input deduction* ($GLI(X)$-deduction, for short) of F from P iff:

1. $F_1 \in P$ or F_1 is the denominator of an instance of a rule R in $\mathcal{L}_{ir}$ such that the numerator of R is empty, and
2. In the sequel, let α be any ordinal and let λ be a limit ordinal.
   (a) $F_{\alpha+1}$ is a generalized resolvent of F_α and some $Y \in X$ where $Y \subseteq \{F_1, F_2, \ldots, F_\alpha\}$ OR
   (b) $F_{\alpha+1}$ is a generalized resolvent of F_α and $\{C\}$ where $C \in P$.
3. If λ is a limit ordinal, then F_λ is a unary resolvent of Y where Y is some subset of $\{F_\gamma \mid \gamma < \lambda\}$.

The above definition is called "linear" because of the similarity with linear resolution in classical theorem proving [26, 27]. Note, in particular, that just as in the case of linear resolution, one parent in each non-unary resolvent must be the resolvent most recently derived (i.e. in order to generate $F_{\alpha+1}$, by non-unary resolution, F_α must be one of the parents).

Theorem 4.1 (Soundness) Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic and $X \subseteq \mathcal{P}(\mathcal{L}_{wff})$. Then: $GLI(X)$-deduction is sound w.r.t. $\mathcal{L}$, i.e. if there is a $GLI(X)$-deduction of formula F from program P (in the abstract logic $\mathcal{L}$), then $P \vdash F$.

Proof. Let $F_0, F_1, F_2, \ldots, F_\alpha$ be a $GLI(X)$-deduction of F from P. We proceed by induction on α.

Base Case. ($\alpha = 0$) Then $F = F_0 = F_\alpha$ is in P and hence, by Definition 2.3, it follows that $P \vdash F$.

Inductive Case. ($\alpha > 1$) Let $\Lambda = \{F_\beta \mid \beta < \alpha\}$. By the induction hypothesis, $P \vdash F_\beta$ for all $\beta < \alpha$. There are two possibilities:

Case 1 (α is a limit ordinal). Then rule 3 in Definition 4.2 must have been used to obtain F_α. Hence, there is a rule in $\mathcal{L}_{ir}$ having an instance of the form:

$$\frac{Numerator}{F_\alpha}$$

such that $Numerator \subseteq \Lambda$. By the induction hypothesis, $P \vdash \Lambda$. As $Numerator \subseteq \Lambda$, $P \vdash Numerator$. Hence, as $P \vdash \Lambda$ and as F_α is obtained by a unary resolvent of some $Y \in X$, and as $Y \subseteq \Lambda$, we know that $P \vdash Y$. By definition of $\vdash$, it now follows that $P \vdash F_\alpha$.

Case 2 (α is a successor ordinal, i.e. $\alpha = (\gamma + 1)$.) In this case, rule (2) in Definition 4.2 must have been used to derive F_α.

If rule 2(a) of Definition 4.2 was used, then F_α is a generalized resolvent of F_γ and Y for some $Y \in X$ and $Y \subseteq \{F_1, \ldots, F_\gamma\}$. By the induction hypothesis, we know that:

$$P \vdash Y \text{ and}$$

$$P \vdash F_\gamma.$$

It follows from Definition 2.3 that $P \vdash F_{\gamma+1} = F_\alpha$.

If rule 2(b) of Definition 4.2 was used, then F_α is a generalized resolvent of F_γ and $\{C\}$ for some $C \in P$. Thus, $P \vdash C$ and, by the induction hypothesis, $P \vdash F_\gamma$. Thus, $P \vdash F_\alpha$. □

One would guess, quite correctly, that $GLI(X)$-deduction is not complete for arbitrary X. We do not know of a necessary condition on X and/or $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ for $GLI(X)$-deduction to be complete, but we do know of a sufficient condition which appears to encompass most existing extensions of logic programming.

Definition 4.3 Suppose $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic and $X \subseteq \mathcal{P}(\mathcal{L}_{wff})$. X is said to possess the *ground inclusion property* for logic $\langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ iff whenever

$$\frac{Numerator}{\Gamma}$$

is an instance of an inference rule of $\mathcal{L}$, then $Numerator \in X$.

Intuitively, $GLI(X)$-deduction allows us to only use those instances of inference rules whose numerators are in X. Thus, even though an inference rule R may be an instance of a rule of $\mathcal{L}_{ir}$ (and hence may be used in derivations), $GLI(X)$-deduction may exclude (or block) the utilisation of rule R in a derivation. Thus, in general, $GLI(X)$-deduction is unlikely to be complete. The ground inclusion property, in essence, says that no instance of an inference rule of $\mathcal{L}_{ir}$ may be excluded by X.

Theorem 4.2 (Completeness) Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is an abstract logic and $X \subseteq \mathcal{P}(\mathcal{L}_{wff})$ possesses the ground inclusion property. Then: $GLI(X)$-deduction is complete w.r.t. $\mathcal{L}$, i.e. if $P \vdash F$, then there is a $GLI(X)$-deduction of formula F from program P.

Proof. The proof follows immediately from the fact that X possesses the ground inclusion property, and hence does not block the use of any inference rule of $\mathcal{L}_{ir}$. □

Note that the above theorem does not say anything about the length of a GLI-deduction. In particular, the reader may recollect that GLI-deductions are allowed to be infinitary. The above completeness theorem does not say anything about the existence of finitary proofs.

Theorem 4.3 (Finitary Completeness) Suppose $\mathcal{L} = \langle \mathcal{L}_\Sigma, \mathcal{L}_{wff}, \mathcal{L}_{mv}, \mathcal{L}_{mt}, \mathcal{L}_{ir} \rangle$ is a finitary abstract logic. Let $X \subseteq \mathcal{P}(\mathcal{L}_{wff})$ have the ground inclusion property. Then: finitary GLI-deduction is complete w.r.t. $\mathcal{L}$, i.e. if $P \vdash F$, then there is a finitary GLI-deduction of formula F from program P.

Proof. The result follows immediately from Theorem 3.2. □

We now revisit the various examples initially described in Section 2 and show how the proof theory described here is applicable to those systems. This, we hope, will explicate the utility of our framework in describing logic programming languages that differ from the traditional definite clause framework in one of two ways:

1. by providing a syntactic variation and/or extension of the definite clause framework
2. by interpreting these programs over various kinds of non-classical logics

We develop below, a couple of examples that show how $GLI(X)$ deduction relates to the operational semantics of some existing logic programming languages.

Example 4.2 (Classical Logic Programming) Consider the example of classical logic (Example 3.1). In classical logic programming, select X to be the set of all clauses expressible using the non-logical symbols of our logic. Then the application of modus ponens yields the unrestricted (cf. Lloyd [25]) SLD-resolution proof procedure which allows ancestry resolution. While this is not quite so powerful as SLD-resolution that forbids ancestry resolution, this presents an overall view about how resolution systems generalize to non-classical logics. The fact that ancestry resolution is not required for definite clauses depends crucially upon the precise syntactic form of the formulas in the definite clause fragment of logic, rather than on the model theoretic properties of first order logic itself (this observation is backed by the fact that SLD-resolution is of course not complete for full first order logic). As X is the set of all clauses in our language, and as any instance of the resolution rule contains only clauses in the numerator, it follows that no rule is "blocked" by X. Recall that unrestricted SLD-resolution is same as SLD-resolution except that arbitrary unifiers, rather than only mgu's, are allowed to participate in the refutation.

Example 4.3 (Multivalued Logic Programming) Consider multivalued logic programming over hte complete lattice $\mathcal{T}$ of truth values using annotated logics (cf. Blair and Subrahmanian [9, 10]). Take X to be the set of all instances of annotated clauses expressible using the non-logical symbols of our language. We have the following inference rules:

$$\frac{\{\mathbf{A} : \mu \mid \mu \in \mathcal{T}'\}}{\mathbf{A} : \bigsqcup_{\mu \in \mathcal{T}'} \mu}.$$

In addition, for each $\mu\rho \in \mathcal{T}$ such that $\mu \leq \rho$, we have the inference rule (called "weakening"):

$$\frac{\mathbf{A} : \rho}{\mathbf{A} : \mu}.$$

Furthermore, we have th familiar rule of modus ponens. Let us see how this relates to the procedure of Blair and Subrahmanian [10]. They show that given any annotated logic program P over $\mathcal{T}$, there is a program $CL(P)$ that can be effectively constructed from P such that P and $CL(P)$ are not only logically equivalent, but they also have the same T_P operator. Suppose we now consider only those programs P such that $P = CL(P)$ (such programs are said to be closed, and by the previous sentence, it follows that there is no loss of generlaity in making this assumption). The nice thing about such programs is that given any annotated atom $A : \mu$, if $CL(P) \models A : \mu$, then there is a *single* clause in P having a ground instance of the form

$$A : \rho \leftarrow Body$$

such that $P \models Body$ and $\rho \geq \mu$. Thus, to show that $P \models A : \mu$, we may look at those clauses in P whose head's atomic part unifies with A and such that the annotation in the head exzceeds μ. However, the application of modus ponens requires that the annotations match precisely. It is to facilitate this that we have the "weakening" rule described above.

Likewise, it is easy to demonstrate that the temporal programs and disjunctive programs described in Section 3 of this paper can also be incorporated into this framework.

5 Discussion

Various kinds of logic programs based on nonclassical logics have been developed. There are now, in particular, quantitative logic programs [39], annotated logic programs [9], three-valued (and n-valued) general logic programs [15], and fuzzy logic programming languages, e.g. PROLOG-ELF [24] and the FPROLOG language of Martin, Baldwin and Pilsworth [28]. The investigations of the quantitative, annotated and fuzzy logic programming languages are motivated, at least in part, by the need for a rigorous semantics for treating uncertainty in deductive databases.

If we look at logic programs of any of the kinds just mentioned for which a definition of *model* has been previously provided we see that the structural characteristics of the class of models of each of these programs are similar. For example, programs without negation have least models in the setting of classical two-valued semantics with respect to any admissible interpretation of their function symbols; in particular they have least Herbrand models.

Similarly, general programs (with negation allowed) have least models with respect to any interpretation of their function symbols in the three-valued semantics. To get even more removed from the classical two-valued setting, consider the quantitative programs of van Emden [39]. Here the underlying truth values are taken to be the interval of real numbers from 0 to 1. As in the ordinary two-valued case we may view programs as operators from interpretations to interpretations and attempt to characterize the models of the programs in terms of their images under these operators. Van Emden sets up the semantics of the rules being considered (which may be described as definite clauses with "attenuation" factors) so that an interpretation I of the program is a model of it iff I is a pre-fixed point of the operator T_P corresponding to the program P. (I is a *pre-fixed point* of T_P if $T_P(I) \leq I$.) Then, by methods that of course specifically depend on the definition of T_P, T_P is shown to be continuous. Once this is done, it can be seen that the class of pre-fixed points of T_P (and fixed points) has many of the same structural properties that it has in the qualitative classical two-valued case.

Guessarian [22] recalls various fixed point theorems in ordered structures and surveys via examples, applications of these theorems in semantics and proof theory, logic programming, and deductive databases. Our aim in this paper has been to expand upon such treatments of logic programming as Guessarian's and show that such theorems are applicable to describing the semantics of logic programs. This semantics is very similar across all of the non-classical logic programming notions that we are examining. In each case we have shown that the semantics that have been provided by the authors of these languages can be obtained by, first, specifying the set of underlying truth values that the logic is intended to have, and second, by defining for each program in the language an operator mapping sets of formulas to sets of formulas. This, we think, illuminates the underlying reasons that unify the similar properties that one sees in the semantics associated with programs of various kinds based upon nonclassical logics.

6 Conclusions

The main aim of this paper has been to present a single uniform framework for studying the semantics of logic programming based on different logics, as well as logic programming with different extensions of classical paradigm. In this paper, we studied only those logics that have a monotone consequence operator. We provided a concept of an abstract logic, and define abstract logic programs in terms of abstract logics. We then derived various results relating fixed-points of operators associated with programs with model-states of the programs. We showed how various systems for paraconsistent logic programming, quantitative logic programming, temporal logic programming, disjunctive logic programming, and distributed logic programming etc. may be captured in our framework. To our knowledge, such a unifying semantical framework for non-classical logic programming and extensions of logic programming had not previously been formulated.

References

[1] M. Abadi and Z. Manna. (1987) *Temporal Logic Programming*, Proc. 4th IEEE Symp. on Logic Programming, pps 4–16, Computer Society Press.

[2] H. Andreka and I. Nemeti. (1978) *The Generalized Completeness of Horn Predicate Logic as a Programming Language*, Acta Cybernetica, 4, pp. 3–10.

[3] K. R. Apt, H. A. Blair and A. Walker. (1988) *Towards a Theory of Declarative Knowledge*, in: Foundations of Deductive Databases and Logic Programming, (ed. Jack Minker), Morgan-Kauffman.

[4] P. Balbiani, L. Farinas del Cerro and A. Herzig. (1988) *Declarative Semantics for Modal Logic Programs*, Rapport Interne No. 292, Languages et Systemes Informatiques, Universite Paul Sabatier, Toulouse Cedex, France.

[5] M. Baudinet. (1989) *Temporal Logic Programming is Complete and Expressive*, Proc. POPL-89.

[6] N.D. Belnap. (1977) *A Useful Four-Valued Logic*, in: Modern Uses of Many-Valued Logic, (eds. G. Epstein and J.M. Dunn), D. Reidel, pps. 8-37.

[7] H. A. Blair. (1982) *The Recursion Theoretic Complexity of the Semantics of Predicate Logic as a Programming Language*, Information and Control, 54, 1/2, pps 25–47.

[8] H.A. Blair. (1986) *Decidability in the Herbrand Base*, in: Proc. of the Workshop on Foundations of Logic Programming and Deductive Databases, (ed. Jack Minker), College Park, MD.

[9] H. A. Blair and V.S. Subrahmanian. (1987) *Paraconsistent Logic Programming*, Proc. 7th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 287, pps 340–360, Springer Verlag. Extended version in: Theoretical Computer Science, Vol. 68, pps 135–154.

[10] H. A. Blair and V. S. Subrahmanian. (1988) *Strong Completeness Results for Paraconsistent Logic Programming*, submitted.

[11] H. A. Blair and V. S. Subrahmanian. (1988) *Paraconsistent Foundations for Logic Programming*, to appear in: Journal of Non-Classical Logic.

[12] L. Farinas del Cerro. (1986) *Molog: A System That Extends Prolog with Modal Logic*, New Generation Computing.

[13] N.C.A. da Costa. (1974) *On the Theory of Inconsistent Formal Systems*, Notre Dame J. of Formal Logic, 15, pps 497-510.

[14] N. C. A. da Costa, V. S. Subrahmanian and C. Vago. (1989) *The Paraconsistent Logics* $\mathbf{P}\mathcal{T}$, to appear in: Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, Vol 37, 1991.

[15] M.C. Fitting. (1985) *A Kripke-Kleene Semantics for Logic Programming*, Journal of Logic Programming, 4, pps 295-312.

[16] M. C. Fitting. (1986) *Partial Models and Logic Programming*, Theoretical Computer Science, 48, pps 229–255.

[17] M. C. Fitting. (1987) *Bilattices and the Theory of Truth*, to appear in: J. of Philosophical Logic.

[18] M. C. Fitting. (1987) *Enumeration Operators and Modular Logic Programming*, Journal of Logic Programming, 4, pps 11–21.

[19] M. C. Fitting. (1988) *Logic Programming on a Topological Bilattice*, Fundamenta Informatica, 11, pps 209–218.

[20] M. C. Fitting. (1988) *Bilattices and the Semantics of Logic Programming*, to appear in: Journal of Logic Programming.

[21] D. M. Gabbay. (1987) *Modal and Temporal Logic Programming*, in: Temporal Logic in Computer Science, (ed. A. Galton), Academic Press.

[22] I. Guessarian. (1987) *Some Fixpoint Techniques in Algebraic Structures and Applications to Computer Science*, Fundamenta Informaticae, 10, 1987, pp. 387-414.

[23] D. Harel. (1979) *First Order Dynamic Logic*, Lecture Notes in Computer Science, Vol. 68, Springer Verlag.

[24] M. Ishizuka and N. Kanai. (1985) *PROLOG-ELF Incorporating Fuzzy Logic*, New Generation Computing, 3, pps 479–486.

[25] J.W. Lloyd. (1984) *Foundations of Logic Programming*, Springer-Verlag.

[26] D. W. Loveland. (1970) *A Linear Format for Resolution*, Proc. IRIA Symp. on Automatic Demonstration, Lecture Notes in Mathematics, Springer, pps 147–162.

[27] D. W. Loveland. (1972) *A Unifying View of some Linear Herbrand Procedures*, JACM, 19, pps 366–384.

[28] T. P. Martin, J. F. Baldwin and B. W. Pilsworth. (1987) *The Implementation of FPROLOG – A Fuzzy Prolog Interpreter*, Fuzzy Sets and Systems, 23, pps 119–129.

[29] J. Minker and A. Rajasekar. (1988) *A Fixpoint Semantics for Non-Horn Logic Programs*, to appear in: Journal of Logic Programming.

[30] L. Naish. (1986) *Negation and Quantifiers in NU-Prolog*, in: Proc. 3rd International Conference on Logic Programming, Lecture Notes in Computer Science, Vol. 225, (ed. E. Shapiro), Springer-Verlag, pps 624-634.

[31] R. Ramanujam and R. K. Shyamsundar. (1984) *Process Specification of Logic Programs*, Proc. 4th FST/TCS Conference, Springer LNCS Vol. 181, pp 31–43.

[32] R. Ramanujam. (1989) *Semantics of Distributed Logic Programs*, Theoretical Computer Science, Vol. 68, pp 203 – 220.

[33] J. A. Robinson. (1965) *A Machine Oriented Logic Based on the Resolution Principle*, Journal of the ACM, 12, pps 23–41.

[34] J. Schoenfield. (1967) *Mathematical Logic*, Addison-Wesley.

[35] E. Shapiro. (1983) *Logic Programs with Uncertainties: A Tool for Implementing Expert Systems*, Proc. IJCAI '83, pps 529–532, William Kauffman.

[36] V. S. Subrahmanian. (1988) *Query Processing in Quantitative Logic Programming*, Proc. 9th International Conference on Automated Deduction, (eds. E. Lusk and R.Overbeek), Lecture Notes in Computer Science Vol. 310, pps 81–100, Springer Verlag.

[37] V. S. Subrahmanian. (1988) *Intuitive Semantics for Quantitative Rule Sets*, Proc. 5th International Conference/Symposium on Logic Programming, (eds. K.Bowen and R.Kowalski), MIT Press, Aug. 1988.

[38] V. S. Subrahmanian. (1988) *Mechanical Proof Procedures for Many Valued Lattice Based Logic Programming*, to appear in: J. of Non-Classical Logic.

[39] M.H. Van Emden. (1986) *Quantitative Deduction and its Fixpoint Theory*, Journal of Logic Programming, 4, 1, pps 37-53.