Antitonic Logic Programs

Carlos Viegas Damásio and Luís Moniz Pereira

Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Universidade Nova de Lisboa
2829-516 Caparica, Portugal. E-mail: cd|lmp@di.fct.unl.pt

Abstract. In a previous work we have defined Monotonic Logic Programs which extend definite logic programming to arbitrary complete lattices of truth-values with an appropriate notion of implication. We have shown elsewhere that this framework is general enough to capture Generalized Annotated Logic Programs, Probabilistic Deductive Databases, Possibilistic Logic Programming, Hybrid Probabilistic Logic Programs and Fuzzy Logic Programming [3, 4]. However, none of these semantics define a form of non-monotonic negation, which is fundamental for several knowledge representation applications. In the spirit of our previous work, we generalise our framework of Monotonic Logic Programs to allow for rules with arbitrary antitonic bodies over general complete lattices, of which normal programs are a special case. We then show that all the standard logic programming theoretical results carry over to Antitonic Logic Programs, defining Stable Model and Well-founded Model alike semantics. We also apply and illustrate our theory to logic programs with costs, extending the original presentation of [17] with a class of negations.

1 Introduction

The generalization of standard logic programming to many-valued logics has been foreseen for a quite long time [20, 8, 9, 14]. Substantial work and results have been attained in the field. In particular, Logic programming literature is prodigal in languages and semantics proposals for extensions of definite logic programs (e.g. [7, 24, 23, 5, 14]), i.e. those without non-monotonic or default negation. Usually, the authors characterize their programs with a model theoretic semantics, where a minimum model is guaranteed to exist, and a corresponding monotonic fixpoint operator (continuous or not).

In a previous work [4], which we recap here, we abstracted out all the details and defined a rather general framework of Monotonic Logic Programs that captures the core of logic programming. In the present paper we generalise the framework to allow for rules with arbitrary antitonic bodies over general complete lattices, of which normal programs are a special case, and show that all the standard logic programming theoretical results carry over to such Antitonic Logic Programs, defining for them Stable and Well-founded Model semantics alike. We also apply and illustrate our theory to logic programs with costs, extending the original presentation of [17] with a class of negations. For this purpose we follow an algebraic approach to the language and the semantics of logic programs, in the same spirit of [6].

Our paper proceeds as follows. First we appeal, for motivation and examples, to the logic programs with costs of [17], where associated with each rule there is a weight or cost factor. In Section 3 we recap Monotonic Logic Programs and supply its theoretical results relevant here. In Section 4, we introduce Antitonic Logic Programs, by means of a transformation into Monotonic programs, and define their semantics, showing they enjoy the properties consonant with a logic programming approach. We end by pointing to future work.

2 Logic Programming with Costs

In [17], rules of definite logic programs are assigned non-negative real numbers, ascribing to every rule in the program the cost of applying it. The authors define three interpretations of cost. The non-reusability approach, which we will discuss in this work, assumes that each individual conclusion about an atom in the program involves a cost. In the reusability approach, we only pay once for concluding about an atom, the first time around. Costs can also be interpreted as time and in this case the semantics proposed is isomorphic to van Emden's quantitative logic programs [22]. Let us illustrate the several approaches with an example:

Example 1. Consider the weighted logic program:

$$a \stackrel{4}{\leftarrow} b, c, c, d \qquad b \stackrel{5}{\leftarrow} \qquad b \stackrel{1}{\leftarrow} c \qquad c \stackrel{3}{\leftarrow} \qquad c \stackrel{0}{\leftarrow} b \quad d \stackrel{6}{\leftarrow}$$

In the non-reusability approach, we spend a minimum of 3 units to conclude c and 4 to conclude b. The minimum cost for a is therefore $4+4+3+3+6=20$.

If we adopt the reusability interpretation, we don't need to pay at all for the use of c at the rule for a, since the cost for c has been paid to conclude b with cost 4. Thus, the cost of atom a is $4+4+6=14$.

Now, for the time interpretation, we assume that the atoms in the body can be proved in parallel. Therefore the cost of applying the rule is the weight of the rule plus the maximum time to derive either atom in the body and there is no reusability. So, the derivation of a takes $4+\max(4,3,3,6)=10$.

In [17] the authors define a model and fixpoint theory for logic programs with costs. We will employ them to illustrate the power of monotonic logic programs, and attain the same results. In the rest of the work we shall focus on much more general logic programs with costs under the non-reusability approach.

3 Monotonic Logic Programs

The theoretical foundations of logic programming were clearly established in [15, 21] for definite logic programs (see also [16]), i.e. programs made up of rules of the form $A_0 \subset A_1 \wedge \ldots \wedge A_n (n \geq 0)$ where each $A_i (0 \leq i \leq n)$ is a propositional symbol (an atom), $\subset$ is classical implication, and $\wedge$ the usual Boolean conjunction.

In this section we generalize the language of definite logic programs in order to encompass more complex bodies and heads and, evidently, many-valued logics. For simplicity, we consider only the propositional (ground) case. Furthermore, we define a model and a fixpoint theory for Monotonic Logic Programs, and extend to them the classical results of logic programming. The important point to realize is that all the fundamental results of logic programming depend only on the monotonicity of the body of the rule and on the fact that it is possible to determine the truth-value of the proposition in the head from the truth-value of the rule body. Our underlying set of truth-values will form a complete lattice and the main idea is that every implication should evaluate to $\top$, the top element in the lattice.

When defining a (new) logic it is necessary to address the two distinct but related aspects: the syntax and the corresponding interpretation of the logical symbols in the language. In this paper we adopt an algebraic characterization of both the language and interpretation of operators. This is a very general and powerful framework, allowing for a simple relation between the two. For lack of space, we reduce the presentation to the essentials. For more details consult for instance [10].

The main assumptions of the paper are collected in the next two definitions.

Definition 1 (Implication Algebra). *Let $\mathfrak{T} =< \mathcal{T}, \preceq >$ be a complete lattice and consider an algebra $\mathfrak{A}$ on the carrier set $\mathcal{T}$. We say that $\mathfrak{A}$ is an implication algebra with respect to $\mathfrak{T}$ iff it has defined an operator $\leftarrow_{\mathfrak{A}}$ on $\mathfrak{A}$ such that*

$$\forall_{a_1, a_2 \in \mathfrak{T}} \ (a_1 \leftarrow_{\mathfrak{A}} a_2) = \top \ \textit{iff} \ a_2 \preceq a_1 \tag{1}$$

where $\top$ is the top element of $\mathfrak{T}$.

Example 2. For logic programs with costs we use as set of truth-values the interval of reals greater or equal than 0, extended with infinity ($[0, \infty]$), ordered by the "greater or equal than" relation. Thus, our bottom element is ∞ and the top element is of course 0. Thus the least upper bound in this complete lattice is the infimum. We designate the complete lattice $< [\infty, 0], \geq >$ by[1] $\mathcal{R}^+$. The implication symbol is defined thus:

$$r_1 \leftarrow_{\mathfrak{C}} r_2 = \begin{cases} 0 & \text{if } r_2 \geq r_1, \text{i.e. } r_1 \leq r_2 \\ \infty & \text{otherwise} \end{cases}$$

This defines an implication algebra $\mathfrak{C} = ([\infty, 0], \leftarrow_{\mathfrak{C}})$.

Notice that some many-valued logics have implication connectives which do not comply with property (1). We refer the reader to [4] for more details.

[1] Note that ∞ in $[\infty, 0]$ is $+\infty$ and not $-\infty$.

Our Monotonic Logic Programs will be constructed from the abstract syntax induced by an implication algebra and a set of propositional symbols. The way syntax and semantics relate in such an algebraic setting is well-known and we defer again to [10] for more details.

Definition 2 (Monotonic Logic Programs). *Let $\mathfrak{A}$ be an implication algebra with respect to a complete lattice $\mathfrak{T}$. Let Π be a set of propositional symbols and $FORM_{\mathfrak{A}}(\Pi)$ the corresponding algebra of formulae freely generated from Π and the "symbols" of operators in $\mathfrak{A}$. A monotonic logic program is a set of rules[2] of the form $A \leftarrow \Psi$ such that:*

1. *The rule $(A \leftarrow \Psi)$ is a formula of $FORM_{\mathfrak{A}}(\Pi)$;*
2. *The head of the rule A is a propositional symbol of Π.*
3. *The body formula Ψ with propositional symbols $B_1, \ldots, B_n$ $(n \geq 0)$ corresponds to an isotonic function having those symbols as arguments.*

As usual, we shall represent binary connectives in infix notation.

A rule of a monotonic logic program expresses a (monotonic) computation method of the truth-value of the head propositional symbol from the truth-values of the symbols in the body. The monotonicity of the rule is guaranteed by the isotonicity of the function corresponding to formula Ψ: if an argument of Ψ is monotonically increased then the truth-value of Ψ also monotonically increases. Notice that the bodies of rules can be formed from any operators in the implication algebra, besides the implication connective.

Example 3. Every rule $A \stackrel{s}{\leftarrow} B_1, \ldots B_m$ of a weighted logic program is translated to the Monotonic Logic Program rule $A \leftarrow s + B_1 + \ldots + B_m$ over the implication algebra $\mathfrak{C}$ extended with the addition operation[3] and with constant symbols for every element of $\mathcal{R}^+$. The program of Example 1 is translated to:

$$a \leftarrow 4 + b + c + c + d \qquad b \leftarrow 5 \qquad b \leftarrow 1 + c \qquad c \leftarrow b \qquad c \leftarrow 3 \qquad d \leftarrow 6$$

Notice that if we decrease (under the ordinary ordering of real numbers) the value of a propositional symbol then the value of the sum also decreases; thus the bodies of the above rules are isotonic with respect to $\mathcal{R}^+$.

An interpretation is simply an assignment of truth-value to each propositional symbol in the language. We assume in the rest of this section an implication algebra $\mathfrak{A}$ with respect to a complete lattice $\mathfrak{T} = <\mathcal{T}, \preceq>$. The operator and implication symbol is denoted by $\leftarrow$. Consider also that a set Π of propositional symbols is given as well as the corresponding algebra $FORM_{\mathfrak{A}}(\Pi)$ of formulae over Π. Then the notion of interpretation is straightforward:

[2] We represent the implication operator $\leftarrow_{\mathfrak{A}}$ at the syntactic level by $\leftarrow$. The same convention applies to any other operator defined in the implication algebra $\mathfrak{A}$.

[3] The sum of ∞ with any other element of $\mathfrak{R}$ renders ∞.

Definition 3 (Interpretation). *An interpretation is a mapping $I : \Pi \to \mathcal{T}$. By the unique homomorphic extension theorem, the interpretation extends uniquely to a valuation function $\hat{I} : FORM_{\mathfrak{A}}(\Pi) \to \mathcal{T}$. The set of all interpretations with respect to the implication algebra $\mathfrak{A}$ is denoted by $\mathcal{I}_{\mathfrak{A}}$.*

The unique homomorphic extension theorem guarantees that for every interpretation of propositional symbols there is an unique associated valuation function. The ordering $\preceq$ on the truth-values in $\mathfrak{T}$ is extended to the set of interpretations as follows:

Definition 4 (Lattice of interpretations). *Consider $\mathcal{I}_{\mathfrak{A}}$ the set of all interpretations with respect to implication algebra $\mathfrak{A}$, and two interpretations $I_1, I_2 \in \mathcal{I}_{\mathfrak{A}}$. Then, $< \mathcal{I}_{\mathfrak{A}}, \sqsubseteq >$ is a complete lattice where $I_1 \sqsubseteq I_2$ iff $\forall_{p \in \Pi}\ I_1(p) \preceq I_2(p)$. The least interpretation $\triangle$ maps every propositional symbol to the least element of $\mathcal{T}$, and the greatest interpretation ∇ maps every propositional symbol to the top element of the complete lattice of truth-values $\mathcal{T}$.*

A rule of a monotonic logic program is satisfied whenever the truth-value of the rule is $\top$. A model is an interpretation which satisfies every rule in the program. Formally:

Definition 5 (Satisfaction and Model). *Consider an interpretation $I \in \mathcal{I}_{\mathfrak{A}}$. A monotonic logic program rule $A \leftarrow \Psi$ is satisfied by I iff $\hat{I}\left((A \leftarrow \Psi)\right) = \top$.*

An interpretation $I \in \mathcal{I}_{\mathfrak{A}}$ is a model of a monotonic logic program P iff all rules in P are satisfied by I.

We proceed by showing that every monotonic logic program has a least model which is the least fixpoint of a monotonic operator, along with other standard logic programming results. One such result is the immediate consequences operator, extending the results of van Emden and Kowalski [21] to the general theoretical setting of implication algebras:

Definition 6 (Immediate consequences operator). *Let P be a monotonic logic program. Define the immediate consequences operator $T_P^{\mathfrak{A}} : \mathcal{I}_{\mathfrak{A}} \to \mathcal{I}_{\mathfrak{A}}$, mapping interpretations to interpretations, where A is a propositional symbol:*

$$T_P^{\mathfrak{A}}(I)(A) = lub \left\{ \hat{I}(\Psi) \text{ such that } A \leftarrow \Psi \in P \right\}$$

The immediate consequences operator evaluates the body of every rule for a propositional symbol A. The truth-value of A is simply the least upper bound of the truth-values of all the bodies of the rules for it.

Example 4. For weighted logic programs the immediate consequences operator reduces to:

$$\begin{aligned} T_P^{\mathfrak{C}}(I)(A) &= lub_{\geq}\ \{s + I(B_1) + \ldots + I(B_m) \mid A \leftarrow s + B_1 + \ldots + B_m \in P\} \\ &= \inf\ \ \{s + I(B_1) + \ldots + I(B_m) \mid A \leftarrow s + B_1 + \ldots + B_m \in P\} \end{aligned}$$

This is very similar to the U_P operator of [17].

A fundamental property of operator $T_P^{\mathfrak{A}}$ is:

Theorem 1 (Monotonicity of the immediate consequences operator). *Let I_1 and I_2 be two interpretations in $\mathcal{I}_{\mathfrak{A}}$, and P a monotonic logic program. Operator $T_P^{\mathfrak{A}}$ is monotonic, i.e. if $I_1 \sqsubseteq I_2$ then $T_P^{\mathfrak{A}}(I_1) \sqsubseteq T_P^{\mathfrak{A}}(I_2)$.*

As usual, the set of models of P is characterized by the post-fixpoints of $T_P^{\mathfrak{R}}$:

Theorem 2. *An interpretation I of $\mathcal{I}_{\mathfrak{A}}$ is a model of a monotonic logic program P iff $T_P^{\mathfrak{A}}(I) \sqsubseteq I$. Thus, the least fixpoint of $T_P^{\mathfrak{A}}$ is the least model of P.*

Therefore, the semantics of a monotonic logic program is given by M_P, the least model of P. One can obtain the least model of a program by transfinite iteration of the immediate consequences operator, as stated in the next theorem:

Theorem 3 (Fixpoint Semantics). *Let P be a monotonic logic program, and consider the transfinite sequence of interpretations of $\mathcal{I}_{\mathfrak{A}}$:*

$$\begin{aligned} T_P^{\mathfrak{A}}{\uparrow}^0 &= \Delta \\ T_P^{\mathfrak{A}}{\uparrow}^{n+1} &= T_P^{\mathfrak{A}}(T_P^{\mathfrak{A}}{\uparrow}^n), \text{ if } n+1 \text{ is a successor ordinal} \\ T_P^{\mathfrak{A}}{\uparrow}^{\alpha} &= \textstyle\bigsqcup_{\beta<\alpha} T_P^{\mathfrak{A}}{\uparrow}^{\beta}, \text{ if } \alpha \text{ is a limit ordinal} \end{aligned}$$

Then, there is an ordinal λ such that $T_P^{\mathfrak{A}}{\uparrow}^{\lambda+1} = T_P^{\mathfrak{A}}{\uparrow}^{\lambda}$, and the least model of P is $M_P = T_P^{\mathfrak{A}}{\uparrow}^{\lambda}$.

The major difference from standard classical logic programming is that our $T_P^{\mathfrak{A}}$ operator might not be continuous, and therefore more than ω iterations may be necessary to "reach" the least fixpoint. All the other important results carry over to our general framework. This possibility is unavoidable if one wants to retain generality. For the study of sufficient conditions to guarantee the continuity of the $T_P^{\mathfrak{A}}$, see [18]. We now determine the least model of the weighted logic program of Example 1:

Example 5. The least interpretation over $\mathfrak{R}$ maps every literal to ∞. Continuing with Example 4, the computation proceeds as follows:

	a	b	c	d
$T_P^{\mathfrak{C}}{\uparrow}^0 =$	∞	∞	∞	∞
$T_P^{\mathfrak{C}}{\uparrow}^1 =$	∞	5	3	6
$T_P^{\mathfrak{C}}{\uparrow}^2 =$	21	4	3	6
$T_P^{\mathfrak{C}}{\uparrow}^3 =$	20	4	3	6
$T_P^{\mathfrak{C}}{\uparrow}^4 =$	20	4	3	6

4 Antitonic Logic Programs

In the preceding section we reviewed the framework of Monotonic Logic Programs. Now we extend the syntax of programs allowing for rules with antitonic bodies using the techniques which have been developed in logic programming theory. Our attained aim is to define well-founded [11] and stable model semantics [12] for logic programming over arbitrary implication algebras. Thus, we can easily extend all the semantics for which we have an embedding into Monotonic Logic Programs with non-monotonic "negations." We apply these new results to logic programs with costs, answering some of the questions raised in [17].

Let us start with the language of Antitonic Logic Programs:

Definition 7 (Antitonic Logic Programs). *Let $\mathfrak{A}$ be an implication algebra with respect to a complete lattice $\mathfrak{T}$. Let Π be a set of propositional symbols and $FORM_{\mathfrak{A}}(\Pi)$ the corresponding algebra of formulae freely generated from Π and the "symbols" of operators in $\mathfrak{A}$. An antitonic logic program is a pair $< P^+, P^- >$ where P^+ and P^- are sets of rules of the form $A \leftarrow \Psi$ such that:*

1. *The rule $(A \leftarrow \Psi)$ is a formula of $FORM_{\mathfrak{A}}(\Pi)$;*
2. *The head of the rule A is a propositional symbol of Π.*
3. *If $A \leftarrow \Psi$ belongs to P^+, then the body formula Ψ with propositional symbols $B_1, \ldots, B_n$ $(n \geq 0)$ corresponds to an isotonic function having those symbols as arguments.*
4. *If $A \leftarrow \Psi$ belongs to P^-, then the body formula Ψ with propositional symbols $B_1, \ldots, B_n$ $(n \geq 0)$ corresponds to an antitonic function having those symbols as arguments.*

We denote a rule of P^+ by $A \overset{+}{\longleftarrow} \Psi$, and a rule of P^- by $A \overset{-}{\longleftarrow} \Psi$.

We recall that a function is isotonic (antitonic) iff the value of the function increases (decreases) when we increase an argument while the other remaining arguments are kept fixed. Note that we do not introduce explicitly a negation symbol in the language. A negation can be obtained by associating an antitonic function with an ordinary propositional symbol, as shown in the next Example.

Example 6. For weighted logic programs we can define a natural class of negations by considering the cost function $\vartheta - x$, onto $\mathcal{R}^+$, which returns 0 if $x \geq \vartheta$, otherwise its value is $\vartheta - x$. The special case of $\infty - x$ returns 0 whenever $x = \infty$, else it evaluates to ∞. The interpretation of this last case is immediate: if an atom cannot be concluded from the program its cost is ∞ and therefore its negation has cost 0. Otherwise, the negation does not hold and therefore the cost of $\infty - x$ is ∞. An example of an antitonic logic program is:

$$
\begin{array}{llll}
a \overset{+}{\longleftarrow} 3 + b + not\_d. & b \overset{+}{\longleftarrow} 2. & d \overset{+}{\longleftarrow} 4 + e. & e \overset{+}{\longleftarrow} 1 + not\_a. \\
not\_a \overset{-}{\longleftarrow} 7 - a & & not\_d \overset{-}{\longleftarrow} 5 - d &
\end{array}
$$

The symbols 'not_a' and 'not_d' are new propositional symbols.

We proceed by first defining a well-founded like semantics for antitonic logic programs, guaranteeing the existence of a model for every program. Afterwards, we deal with stable models, which are simpler to introduce. Again, we assume an implication algebra $\mathfrak{A}$ with respect to a complete lattice $\mathfrak{T} =< \mathcal{T}, \preceq >$. Consider too that a set Π of propositional symbols is given as well as the corresponding algebra $FORM_{\mathfrak{A}}(\Pi)$ of formulae over Π. To start with a new notion of interpretation is required:

Definition 8 (Partial Intepretations). *A partial interpretation is a pair of interpretations* $< I^t, I^{tu} >$*. The set of all partial interpretations is* $\mathcal{I}^p_{\mathfrak{A}}$*.*

The I^t component contains what is "true" in the interpretation, while I^{tu} what is "non-false", i.e. true or undefined. It is important to mark that we do not impose consistency of the interpretation (i.e. $I^t \sqsubseteq I^{tu}$) and thus allow for paraconsistency. Two orders among partial interpretations are useful:

Definition 9 (Truth and knowledge ordering). *Let* I_1 *and* I_2 *be two partial interpretations. The truth and knowledge orders among partial interpretations are defined by:*

Truth ordering: $I_1 \sqsubseteq_t I_2$ *iff* $I_1^t \sqsubseteq I_2^t$ *and* $I_1^{tu} \sqsubseteq I_2^{tu}$*.*
Knowledge ordering: $I_1 \sqsubseteq_k I_2$ *iff* $I_1^t \sqsubseteq I_2^t$ *and* $I_2^{tu} \sqsubseteq I_1^{tu}$*.*

The truth ordering prefers partial interpretations with propositional symbols having higher true truth-value and lesser false truth-value. The knowledge ordering, prefers interpretations with less undefinedness.

The set of partial interpretations ordered by $\sqsubseteq_t$ *or* $\sqsubseteq_k$ *is a complete lattice. Clearly, the bottom and top elements of these lattices are* $\perp_t = <\Delta, \Delta>$*,* $\top_t = <\nabla, \nabla>$*,* $\perp_k = <\Delta, \nabla>$ *and* $\top_k = <\nabla, \Delta>$*.*

Given a partial interpretation, we can define the model of an antitonic logic program:

Definition 10. *Let* $P = < P^+, P^- >$ *be an antitonic logic program. A partial interpretation* $I = < I^t, I^{tu} >$ *satisfies a rule* $A \leftarrow \Psi$ *of* P *iff:*

- $I^t(A) \leftarrow_{\mathfrak{A}} \hat{I}^t(\Psi) = \top$*, if the rule belongs to* P^+*.*
- $I^t(A) \leftarrow_{\mathfrak{A}} \hat{I}^{tu}(\Psi) = \top$*, if the rule belongs to* P^-*.*

We say that I *is a model of* P *iff the interpretation satisfies all the rules of* P*.*

Notice that the heads of rules are always evaluated with respect to I^t while the truth-values of bodies are determined from I^t (I^{tu}) for isotonic (antitonic) rules. The attentive reader will have noticed that with the above definition our programs define only positive information from the bodies. Because of the antitonic character of rules in P^-, the greater the body arguments the lesser is $\hat{I}^{tu}(\Psi)$ and the lesser is $I^t(A)$. Clearly, our framework captures normal logic programs: every rule of the form $A \leftarrow B_1, \ldots, B_m, \mathrm{not}C_1, \ldots, \mathrm{not}C_n$ is translated into $A \leftarrow B_1 \wedge \ldots \wedge B_m \wedge \mathrm{not\_}C_1 \wedge \ldots \wedge \mathrm{not\_}C_n$ and a further rule exists for each new atom $\mathrm{not\_}C_i$ of the form $\mathrm{not\_}C_i \leftarrow 1 - C_i$. The underlying lattice of truth-values is $\{0, 1\}$ with $0 < 1$.

A natural generalization is to include rules which declare directly the truth-values of I^{tu}. Here we enter the arena of extending logic programming and general logic programming, where rules with explicit and default negation in their heads are permitted, which we will deal with in subsequent work.

In order to define the semantics of Antitonic Logic Programs we will resort to a Gelfond-Lifschitz like division operator [12], which transforms an Antitonic

Logic Program into a Monotonic one, for which we know how to compute the corresponding least model. This is the usual technique for defining well-founded and stable model semantics for normal logic programs. Our definition and results orbit around this notion.

Definition 11 (Program Division). *Consider an antitonic logic program P and an interpretation*[4] *I^{tu}. The division of program P by I^{tu} is the monotonic logic program:*

$$\frac{P}{I^{tu}} = P^{+} \cup \left\{ A \leftarrow \vartheta \text{ such that } A \leftarrow \Psi \in P^{-} \text{ and } \vartheta = \hat{I}^{tu}(\Psi) \right\}$$

We assume that every truth-value in $\mathfrak{T}$ has a corresponding constant operator in $\mathfrak{A}$ which returns its value.

Example 7. Consider the program of Example 6 and interpretation I mapping a, not_a, b, d, not_d and f to ∞, 0, 2, 3, 0 and 1, respectively. The division of that program by I is:

$$\begin{array}{llll} a \leftarrow 3 + b + not\_d & b \leftarrow 2 & d \leftarrow 4 + e & e \leftarrow 1 + not\_a \\ not\_a \leftarrow 0 & & not\_d \leftarrow 2 & \end{array}$$

Thus, the program division operation substitutes the bodies of antitonic rules by their truth-value in the given interpretation. The result is a monotonic logic program. Now we can define an auxiliary operator mapping partial interpretations into interpretations, which will be extensively used:

Definition 12. *Consider an antitonic logic program P and a partial interpretation $I =< I^t, I^{tu} >$. Then, $C_P^{\mathfrak{A}}(I) = T^{\mathfrak{A}}_{\frac{P}{I^{tu}}}(I^t)$.*

The $C_P^{\mathfrak{A}}(I)$ operator determines what follows immediately from the program given the partial interpretation. We have the following important results:

Proposition 1. *A partial interpretation I is a model of an antitonic logic program P iff $C_P^{\mathfrak{A}}(I) \sqsubseteq I^t$. Furthermore, let J be another partial interpretation such that $I \sqsubseteq_k J$ then $C_P^{\mathfrak{A}}(I) \sqsubseteq C_P^{\mathfrak{A}}(J)$.*

The last proposition is simply assuring that if we have more knowledge (less undefined values in the interpretation) then we can extract more information from our programs.

Operator $C_P^{\mathfrak{A}}$ maps partial interpretations to interpretations. We now define a new operator which maps partial interpretations to partial interpretations, given what is known to be true and to be false, in the same spirit of Przymusinski's Θ operator [19]:

---

[4] Mark well this is an interpretation, not a partial interpretation!

Definition 13 (Partial Consequences Operator). *Let P be an antitonic logic program, and the two partial interpretations I and J. The partial consequences operator is given by the equation:*

$$\Theta_P^{\mathfrak{A}}(I,J) = \left\langle C_P^{\mathfrak{A}}\left(< I^t, J^{tu} >\right), C_P^{\mathfrak{A}}\left(< I^{tu}, J^t >\right)\right\rangle$$

We usually omit the subscript $\mathfrak{A}$ and denote $\Theta_P^{\mathfrak{A}}(I,J)$ by $\Theta_P^J(I)$.

Interpretation J represents safe knowledge. On the one hand, from J^{tu} one can conclude what definitely doesn't hold, and thus we use this interpretation for deriving what holds surely. On the other hand, from J^t one knows what is true, and so we use it for deriving what may possibly hold.

Proposition 2. *Let I, J and K be partial interpretations. If $I \sqsubseteq_t J$ then $\Theta_P^K(I) \sqsubseteq_t \Theta_P^K(J)$.*

Basically we have shown in the previous proposition that Θ_P^J is a monotonic operator on the lattice of partial interpretations according to the truth-ordering. Its least fixpoint is guaranteed to exist and is important for our objectives:

Definition 14 (Full Consequences Operator). *Let P be an antitonic logic program and J a partial interpretation. Define $\Omega_P^{\mathfrak{A}}(J) = \text{lfp } \Theta_P^J$. Alternatively, $\Omega_P^{\mathfrak{A}}(J)$ is given by $\Theta_P^J{\uparrow}^\lambda$ where λ is the least ordinal for which $\Theta_P^J{\uparrow}^{\lambda+1} = \Theta_P^J{\uparrow}^\lambda$ with*

$$\begin{aligned}
\Theta_P^J{\uparrow}^0 &= <\Delta, \Delta> \\
\Theta_P^J{\uparrow}^{n+1} &= \Theta_P^J(\Theta_P^J{\uparrow}^n), \textit{ if } n+1 \textit{ is a successor ordinal} \\
\Theta_P^J{\uparrow}^\alpha &= \textstyle\bigsqcup_t \left\{\Theta_P^J{\uparrow}^\beta \textit{ such that } \beta < \alpha\right\}, \textit{ if } \alpha \textit{ is a limit ordinal}
\end{aligned}$$

By an easy transfinite induction proof one concludes:

Proposition 3. *Let P be an antitonic logic program and J an arbitrary partial interpretation then:*

$$\Theta_P^J{\uparrow}^\alpha = \left\langle T^{\mathfrak{A}}_{\frac{P}{J^{tu}}}{\uparrow}^\alpha, T^{\mathfrak{A}}_{\frac{P}{J^t}}{\uparrow}^\alpha\right\rangle$$

So, the least fixpoint of Θ_P^J is in fact obtained from the least model of monotonic logic programs $\frac{P}{J^{tu}}$ and $\frac{P}{J^t}$. This deserves a definition and a theorem:

Definition 15 (Gamma operator). *Let P be an antitonic logic program and J an interpretation. Define*

$$\Gamma_P^{\mathfrak{A}}(J) = M_{\frac{P}{J}} = \text{lfp } T^{\mathfrak{A}}_{\frac{P}{J}} = T^{\mathfrak{A}}_{\frac{P}{J}}{\uparrow}^\lambda, \textit{ for some ordinal } \lambda$$

Theorem 4. *Let P be an antitonic logic program and J a partial interpretation. Then,*

$$\Omega_P^{\mathfrak{A}}(J) = \left\langle \Gamma_P^{\mathfrak{A}}(J^{tu}), \Gamma_P^{\mathfrak{A}}(J^t)\right\rangle$$

Thus, we have defined the full consequences operator in terms of the least model of two monotonic logic programs. Basically, Theorem 4 states that the least model of $\frac{P}{J^{tu}}$ gives what is true given the safe knowledge J, and that the least model of $\frac{P}{J^t}$ provides what is non-false.

A well-known result of logic programming theory is that operator Γ is anti-monotonic. The same happens with antitonic logic programs:

Proposition 4. *Consider the antitonic logic program P. Let I and J be two interpretations such that $I \sqsubseteq J$, then $\Gamma_P^{\mathfrak{A}}(J) \sqsubseteq \Gamma_P^{\mathfrak{A}}(I)$.*

From this there follows a fundamental result:

Theorem 5 (Monotonicity of the full consequences operator). *Let P be an antitonic logic program, and $I \sqsubseteq_k J$ two partial interpretations. Then, $\Omega_P^{\mathfrak{A}}(I) \sqsubseteq_k \Omega_P^{\mathfrak{A}}(J)$.*

We conclude immediately, again by the Knaster-Tarski fixpoint theorem, that $\Omega_P^{\mathfrak{A}}$ has a least fixpoint under the knowledge ordering of partial interpretations.

Definition 16 (Well-founded Semantics). *Consider an antitonic logic program P. The partial stable models of P are the fixpoints of operator $\Omega_P^{\mathfrak{A}}$. The least one under the knowledge ordering of partial interpretations is the well-founded model WFM_P, and can be obtained by transfinitely iterating the $\Omega_P^{\mathfrak{A}}$ operator from the least partial interpretation $\bot_k = <\Delta, \nabla>$.*

Given a partial stable model M we say that an atom A is:

- **true** *with degree ϑ wrt. M if $\vartheta \preceq M^t(A)$ and $\vartheta \preceq M^{tu}(A)$.*
- **undefined** *with degree ϑ wrt. M if $\vartheta \not\preceq M^t(A)$ and $\vartheta \preceq M^{tu}(A)$.*
- **false** *with degree ϑ wrt. M if $\vartheta \not\preceq M^t(A)$ and $\vartheta \not\preceq M^{tu}(A)$.*
- **inconsistent** *with degree ϑ wrt. M if $\vartheta \preceq M^t(A)$ and $\vartheta \not\preceq M^{tu}(A)$.*

By resorting to the definition of $\Omega_P^{\mathfrak{A}}$ in terms of $\Gamma_P^{\mathfrak{A}}$ we can present an alternating fixpoint definition [1] of the well-founded semantics for antitonic logic programs:

Theorem 6. *Consider an antitonic logic program P. The partial interpretation I is a fixpoint of $\Omega_P^{\mathfrak{A}}$ iff $\Gamma_P^{\mathfrak{A}}(\Gamma_P^{\mathfrak{A}}(I^t)) = I^t$ and $I^{tu} = \Gamma_P^{\mathfrak{A}}(I^t)$.*

Example 8. The well-founded model in Example 6 is determined as follows:

	a	not_a	b	d	not_d	e
$I_0 =$	∞	∞	∞	∞	∞	∞
$\Gamma_P^{\mathfrak{C}}(I_0) =$	5	0	2	5	0	1
$I_1 = \Gamma_P^{\mathfrak{C}}(\Gamma_P^{\mathfrak{C}}(I_0)) =$	5	2	2	7	0	3
$\Gamma_P^{\mathfrak{C}}(I_1) =$	5	2	2	7	0	3

The well-founded model of P is $< I_1, \Gamma_P^{\mathfrak{C}}(I_1) >=< I_1, I_1 >$, and thus we conclude a with cost 5. The example illustrates a possible interpretation of negation in logic programs with costs: what costs more to prove true costs less to show false, and vice-versa.

Thus, the well-founded model of P is given by the least fixpoint of the $\left(\Gamma_P^{\mathfrak{A}}\right)^2$ operator, a well-known result in the theory of logic programming [1]. We have the next reassuring result:

Theorem 7. *The partial stable models of an antitonic logic program P are models of P.*

A partial stable model is said consistent whenever there is no propositional symbol A which is inconsistent to any degree ϑ. It is said fully defined whenever there is no propositional symbol with undefined truth-value to any degree ϑ. Clearly, the following is expected:

Proposition 5. *A partial stable model $\mathcal{M}$ of antitonic logic program P is consistent iff $\mathcal{M}^t \sqsubseteq \mathcal{M}^{tu}$. It is fully defined whenever $\mathcal{M}^{tu} \sqsubseteq \mathcal{M}^t$.*

The consistent and fully-defined partial stable models of P are the stable models of P:

Definition 17 (Stable Models). *Let P be an antitonic logic program. We say that M is a stable model of P iff it is a consistent and fully defined partial stable model of P. Furthermore, $M^t = M^{tu}$ and M is a stable model iff it is a fixpoint of $\Gamma_P^{\mathfrak{A}}$, i.e. $M = \Gamma_P^{\mathfrak{A}}(M)$.*

The program of Example 6 has a single stable model which coincides with its well-founded model.

Example 9. The program containing the rules $a \overset{-}{\longleftarrow} \vartheta_1 - not\_a.$ and $not\_a \overset{-}{\longleftarrow} \vartheta_2 - a.$ has a single stable model when $\vartheta_1 = 0$ or $\vartheta_2 = 0$. Suppose now that both are finite non-zero real numbers. In this case if $\vartheta_1 \neq \vartheta_2$ then the program has no stable models. Assume that $\vartheta_1 = \vartheta_2 > 0$. All interpretations such that $a + not\_a = \vartheta_1$ are stable models of the program, therefore there exist an infinite number of them. Finally, if both $\vartheta_1 = \vartheta_2 = \infty$ then there are two stable models of the program. If $\vartheta_1 = \infty$ and $\vartheta_2 \neq \infty$ we have a single stable model.

The analysis of the program comprising the single rule $a \overset{-}{\longleftarrow} \vartheta - a$ is easier. If $\vartheta = \infty$ then there are no stable models. Otherwise, the single stable model assigns the truth-value $\frac{\vartheta}{2}$ to a.

Example 10. Consider this variant of Example 6:

$$
\begin{array}{llll}
a \overset{+}{\longleftarrow} 3 + b + not\_d. & b \overset{+}{\longleftarrow} 2. & d \overset{+}{\longleftarrow} 4 + e. & e \overset{+}{\longleftarrow} 1 + not\_a. \\
not\_a \overset{-}{\longleftarrow} \infty - a & & not\_d \overset{-}{\longleftarrow} \infty - d &
\end{array}
$$

The well-founded model and its two stable models are shown below :

	a	not_a	b	d	not_d	e
$WFM^t =$	∞	∞	2	∞	∞	∞
$WFM^{tu} =$	5	0	2	5	0	1

	a	not_a	b	d	not_d	e
$M_1 =$	∞	0	2	5	∞	1
$M_2 =$	5	∞	2	∞	0	∞

The positive part of the well-founded model is not very informative, except that we can conclude b with cost 2. However, the non-false part provides the joint limits of the stable models. In general, $WFM^t \sqsubseteq \sqcap_i M_i$ and $\sqcup_i M_i \sqsubseteq WFM^{tu}$.

The interpretation of the negation $\infty - A$ in stable model semantics should now be clear. Given an atom that we can prove then its negation is not provable and it therefore costs ∞ to prove. Also, by simply looking at the model, one can see what does not hold, and therefore know instantly that its negation holds without any extra effort, justifying its zero cost. We believe this is in accordance with the ideas advanced in the conclusions section of [17], as a possible interpretation for negation in logic programs with costs. Obviously, any other antitonic negation can be easily encoded in our framework.

5 Conclusions and Future Work

In a single sentence, this paper shows how to (logically) program with arbitrary monotonic and anti-monotonic (antitonic) operators over a complete lattice. This is a simple and powerful idea with an enormous range of applications. In particular, the present work paves the way to combine and integrate several forms of reasoning into a single framework, namely fuzzy, probabilistic, uncertain, and paraconsistent ones.

This paper is the natural extension of the seminal works by Subrahmanian [20, 14] and Fitting [8, 9]. However, these authors stick to a logic programming syntax instead of considering arbitrary monotonic and antitonic functions in the bodies. This is a major contribution of our work. To be absolutely fair, the article [9] introduces the notions of *attenuation operators* which can be viewed as arbitrary monotonic functions over bilattices.

Quite recently, there appeared in the literature a profound work [6] entitled "Approximations, Stable Operators, Well-founded Fixpoints and Applications in Nonmonotonic Reasoning". The stance there is to depart from either a monotone or antitonic operator and define approximations to it, via a *stable operator* which corresponds to an abstract version of the well-founded semantics. The interesting case is when O is antitonic. In this situation, most of the results in [6] are obtained in our scheme by considering the single rule program $A \overset{-}{\leftarrow} O(A)$. It is also true that our results follow immediately from [6]. So, both frameworks are capable of expressing each other. However, we show how multiple operators should be combined in order to extend the widely accepted well-founded and stable model semantics for normal logic programs, a point not addressed in [6].

Regarding future work, we expect to show the embeddings of other cost functions introduced in [17], as well as those of [22]. Extension of our antitonic results to residuated lattices (c.f. [2, 13]), where a generalized modus ponens rule is defined, is also foreseen. We also aim to provide a proof theory for (instances) of our framework.

References

1. C. Baral and V. S. Subrahmanian. Duality between alternative semantics of logic programs and nonmonotonic formalisms. *Journal of Automated Reasoning*, 10:399–420, 1993.

2. L. Bolc and P. Borowik. *Many-Valued Logics. Theoretical Foundations.* Springer–Verlag, 1992.
3. C. V. Damásio and L. M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. In M. O. Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira, editors, *Proc. of JELIA'00*, pages 57–72. LNAI 1919, Springer–Verlag, 2000.
4. C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs, 2001. Accepted in ECSQARU-2001.
5. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. In *International Conference on Logic Programming 1997*, pages 391–495. MIT Press, 1997.
6. M. Denecker, V. Marek, and M. Truszczyński. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In J. Minker, editor, *Logic-Based Artifical Intelligence*, pages 127–144. Kluwer Academic Publishers, 2000.
7. D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In *International Conference on Logic Programming 1991*, pages 581–598. MIT Press, 1991.
8. M. Fitting. Bilattices and the semantics of logic programs. *Journal of Logic Programming*, 11:91–116, 1991.
9. M. Fitting. The family of stable models. *Journal of Logic Programming*, 17:197–225, 1993.
10. J. H. Gallier. *Logic for Computer Science.* John Wiley & Sons, 1987.
11. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
12. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
13. P. Hájek. *Metamathematics of Fuzzy Logic.* Trends in Logic. Studia Logica Library. Kluwer Academic Publishers, 1998.
14. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
15. R. Kowalski. Predicate logic as a programming language. In *Proceedings of IFIP'74*, pages 569–574. North Holland Publishing Company, 1974.
16. J. W. Lloyd. *Foundations of Logic Programming.* Springer–Verlag, 1987.
17. V. W. Marek and M. Truszczyński. Logic programming with costs. Technical report, 2000. Available at ftp://al.cs.engr.uky.edu/cs/manuscripts/lp-costs.ps.
18. J. Medina, M. Ojeda-Aciego, and P. Vojtas. Multi-adjoint logic programming with continuous semantics. In *Proc. of LPNMR'01*, September 2001. This issue.
19. H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence, a Sourcebook*, pages 321–367. North Holland, 1990.
20. V.S. Subrahmanian. Algebraic properties of the space of multivalued and paraconsistent logic programs. In *9th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 56–67. LNCS 405, Springer–Verlag, 1989.
21. M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.
22. M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4:37–53, 1986.
23. P. Vojtas. Fuzzy logic programming. *Fuzzy Sets and Systems*, 2001. Accepted.
24. P. Vojtás and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Proc. of the Ws. on Extensions of Logic Programming (ELP'96)*, pages 289–301. LNCS 1050, Springer–Verlag., 1996.