The Well-founded Semantics in Normal Logic Programs with Uncertainty

Yann Loyer and Umberto Straccia

Istituto di Elaborazione della Informazione -C.N.R.
Via G. Moruzzi,1 I-56124 Pisa (PI) ITALY

Abstract. Many frameworks of logic programming have been proposed to manage uncertain information in deductive databases and expert systems. Roughly, on the basis of how uncertainty is associated to facts and the rules in a program, they can be classified into *implication-based* (IB) and *annotation-based* (AB). However, one fundamental issue that remains unaddressed in the IB approach is the representation and the manipulation of the non-monotonic mode of negation, an important feature for real applications. Our focus in this paper is to introduce non-monotonic negation in the *parametric* IB framework, a unifying umbrella for IB frameworks. The semantical approach that we will adopt is based on the well-founded semantics, one of the most widely studied and used semantics of (classical) logic programs with negation.

1 Introduction

The management of uncertainty within deduction systems is an important issue in all those AI application domains in which the real world information to be represented is of imperfect nature (which is likely the rule rather than an exception). An impressive work has been carried out in the last decades, resulting in a number of concepts being investigated, a number of problems being identified and a number of solutions being developed (see, *e.g.* [1, 6, 17, 29]).

First-Order Logic (FOL) has been the basis for most knowledge representation formalisms. Its basic units –individuals, their properties, and the relationship between them– naturally capture the way in which people encode their knowledge. Unfortunately, it is severely limited in its ability to represent our uncertainty about the world: a fact can only be known to be true, known to be false or neither. By contrast, most of our knowledge about the real world is not absolutely true. Additionally, practical considerations dictate that the framework used for knowledge representation with uncertainty admit efficient implementation and efficient computations. Logic database programming, with its advantage of modularity and its powerful top-down and bottom-up query processing techniques, has attracted the attention of researchers and numerous frameworks for deductive databases with uncertainty have been proposed [2, 4, 5, 9, 10, 14–16, 18–22, 25–27, 35–39], where the underlying uncertainty formalism include probability theory [10, 19, 22, 25–27, 39], fuzzy set theory [2, 35, 37, 38], multi-valued logic [9, 15, 16, 20, 21] and possibilistic logic [5]. Roughly, based on

the way in which uncertainty is associated with the facts and rules of a program, these frameworks can be classified into *annotation based* (AB) and *implication based* (IB).

In the AB approach (se *e.g.* [15, 16, 25–27, 36]), a rule is of the form

$$A : f(\beta_1, \ldots, \beta_n) \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n$$

which asserts "the certainty of the atom A is at least (or is in) $f(\beta_1, \ldots, \beta_n)$, whenever the certainty of the atom B_i is at least (or is in) β_i, $1 \leq i \leq n$", where f is an n-ary computable function and β_i is either a constant or a variable ranging over an appropriate certainty domain.

On the other hand, in the IB approach (see *e.g.* [7, 9, 18–20, 37]), a rule is of the form

$$A \stackrel{\alpha}{\leftarrow} B_1, ..., B_n$$

which says that the certainty associated with the implication $B_1 \wedge ... \wedge B_n \rightarrow A$ is α. Computationally, given an assignment I of certainties to the B_is, the certainty of A is computed by taking the "conjunction" of the certainties $I(B_i)$ and then somehow "propagating" it to the rule head. It is not our aim to compare the two approaches in this paper. Refer to [20] for an exhaustive comparison. We limit our contribution in this sense to recall the following facts [20]: (*i*) while the way implication is treated in the AB approach is closer to classical logic, the way rules are fired in the IB approach has a definite intuitive appeal and (*ii*) the AB approach is strictly more expressive than the IB. The down side is that query processing in the AB approach is more complicated, *e.g.* the fixpoint operator is not continuous in general, while it is in the IB approaches. From the above points, it is believed that the IB approach is easier to use and is more amenable for efficient implementation.

However, one fundamental issue that remains still unaddressed in the IB approach is the representation and the manipulation of the non-monotonic mode of *default negation*, *not* A, an without doubts important feature to be used in applications. The major distinctive feature is that *not* A is assumed in the absence of sufficient evidence to the contrary. The meaning of "sufficient evidence" depends on the specific semantics used. Due to its importance, the problem of negation in logic programs has attracted many researchers and a broad variety of semantical approaches have been invented. For example, in Reiter's *Closed World Assumption* [34], *not* A is assumed for atomic A if A is not provable, or, equivalently, if there is a minimal model in which A is false. On the other hand, in Minker's *Generalised Closed World Assumption* [24, 13], or in McCarthy's *Circumscription*, [23], *not* A is assumed only if A is false in *all* minimal models. In Clark's *Predicate Completion* semantics for logic programs [3] this form of negation is called *negation-as-failure* because *not* A is derived whenever attempts to prove A finitely fail. The more recent semantics proposed for logic programs and deductive databases, such as the *stable semantics* [12], *well-founded semantics* [28], *partial stable* or *stationary semantics* [33], and *static semantics* [30], propose even more sophisticated meanings for default negation.

Contributions: We will extend the *parametric* IB framework [20], a unifying umbrella for IB frameworks, with default negation. The semantical approach that we will adopt is based on the well-founded semantics, one of the most widely studied and used semantics of (classical) logic programs with negation. From a semantics point of view, we will combine an alternating fixpoint semantics similar to [11] with the fixpoint characterisation of [20]. We will show that in case of positive programs, the *parametric* IB framework is obtained, while restricting logic programs to Datalog programs, the classical well-founded semantics is obtained, *i.e.* our extension is a conservative extension.

The use of default negation has already been considered in some deductive databases with uncertainty frameworks. For instance, in [25], the stable semantics has been considered within the AB approach, but limited to the case where the underlying uncertainty formalism is probability theory. The stable semantics has been considered also in [38], where a semi-possibilistic logic has been proposed. In it, a particular negation operator has been introduced and a fixed min/max-evaluation of conjunction and disjunction is adopted. To the best of our knowledge, there is no work dealing with default negation within the parametric IB approach.

The remaining part of the paper is organized as follows. The syntax of programs, called normal parametric programs, is given in Section 2. In Section 3 the notions of interpretation and model of a program are defined, while Section 4, we define the intended model of a normal parametric program and show that our semantics extends the well-founded semantics [28] and the Shiri-Lakshmanan's semantics [20] to normal parametric programs. Section 5 contains concluding remarks.

2 Preliminaries

We recall the syntactical aspects of the parametric IB framework presented in [20] and extend it with negation.

Consider an arbitrary first order language that contains infinitely many variable symbols, finitely many constants, and predicate symbols, but no function symbols. The predicate symbol $\pi(A)$ of an atomic formula A given by $A = p(X_1, \ldots, X_n)$ is defined by $\pi(A) = p$. While the language does not contain function symbols, it contains symbols for families of propagation ($\mathcal{F}_p$), conjunction ($\mathcal{F}_c$) and disjunction functions ($\mathcal{F}_d$), called *combination functions*.

Let $\mathcal{L} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$ be a certainty lattice (a complete lattice), where $\mathcal{T}$ is a set of truth values, and let $\mathcal{B}(\mathcal{T})$ the set of finite multisets over $\mathcal{T}$ (multisets are indicated with $\{\!\{ \cdot \}\!\}$). With $\bot$ and $\top$ we denote the least and greatest element in $\mathcal{T}$, respectively. A *propagation function* is a mapping from $\mathcal{T} \times \mathcal{T}$ to $\mathcal{T}$ and a *conjunction* or *disjunction* function is a mapping from $\mathcal{B}(\mathcal{T})$ to $\mathcal{T}$. Each kind of function must verify some of the following properties[1]:

[1] For simplicity, we formulate the properties treating any function as a binary function on $\mathcal{T}$.

1. monotonicity w.r.t. (with respect to) each one of its arguments;
2. continuity w.r.t. each one of its arguments;
3. bounded-above: $f(\alpha_1, \alpha_2) \preceq \alpha_i$, for $i = 1, 2, \forall \alpha_1, \alpha_2 \in \mathcal{T}$;
4. bounded-below: $f(\alpha_1, \alpha_2) \succeq \alpha_i$, for $i = 1, 2, \forall \alpha_1, \alpha_2 \in \mathcal{T}$;
5. commutativity: $f(\alpha_1, \alpha_2) = f(\alpha_2, \alpha_1), \forall \alpha_1, \alpha_2 \in \mathcal{T}$;
6. associativity: $f(\alpha_1, f(\alpha_2, \alpha_3)) = f(f(\alpha_1, \alpha_2), \alpha_3), \forall \alpha_1, \alpha_2, \alpha_3 \in \mathcal{T}$;
7. $f(\{\alpha\}) = \alpha, \forall \alpha \in \mathcal{T}$;
8. $f(\emptyset) = \bot$;
9. $f(\emptyset) = \top$;
10. $f(\alpha, \top) = \alpha, \forall \alpha \in \mathcal{T}$;

As postulated in [20]:

1. a conjunction function in $\mathcal{F}_c$ should satisfy properties 1, 2, 3, 5, 6, 7, 9 and 10;
2. a propagation function in $\mathcal{F}_p$ should satisfy properties 1, 2, 3 and 10;
3. a disjunction function in $\mathcal{F}_d$ should satisfy properties 1, 2, 4, 5, 6, 7 and 8.

We also assume that there is a function from $\mathcal{T}$ to $\mathcal{T}$, called *negation function* and denoted $\neg$, that is anti-monotone w.r.t. $\preceq$ and satisfies $\neg\neg\alpha = \alpha, \forall \alpha \in \mathcal{T}$ and $\neg\bot = \top$.

Definition 1 (Normal parametric program). *A* normal parametric program *P (np-program) is a 5-tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{C}, \mathcal{P}, \mathcal{D} \rangle$, whose components are defined as follows:*

1. *$\mathcal{L} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$ is a complete lattice, where $\mathcal{T}$ is a set of truth-values partially ordered by $\preceq$, $\otimes$ is the meet operator and $\oplus$ the join operator. We denote the least element of the lattice by $\bot$ and the greatest element by $\top$;*
2. *$\mathcal{R}$ is a finite set of* normal parametric rules *(np-rules), each of which is a statement of the form:*

$$r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n, \neg C_1, ..., \neg C_m$$

   where A is an atomic formula and $B_1, ..., B_n, C_1, ..., C_m$ are atomic formulas or values in $\mathcal{T}$ and $\alpha_r \in \mathcal{T} \setminus \{\bot\}$ is the certainty of the rule;
3. *$\mathcal{C}$ is a mapping that associates with each np-rule a conjunction function in $\mathcal{F}_c$;*
4. *$\mathcal{P}$ is a mapping that associates with each np-rule a propagation function in $\mathcal{F}_p$;*
5. *$\mathcal{D}$ is a mapping that associates with each predicate symbol in P a disjunction function in $\mathcal{F}_d$.* ∎

For ease of presentation, we write

$$r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n, \neg C_1, ..., \neg C_m; \langle f_d, f_p, f_c \rangle$$

to represent a np-rule in which $f_d \in \mathcal{F}_d$ is the disjunction function associated with the predicate symbol $\pi(A)$ of A and, $f_c \in \mathcal{F}_c$ and $f_p \in \mathcal{F}_p$ are respectively the conjunction and propagation functions associated with r. The intention is that the conjunction function (*e.g.* $\otimes$) determines the truth value of the conjunction

of $B_1, ..., B_n, \neg C_1, ..., \neg C_m$, the propagation function (*e.g.* $\otimes$) determines how to "propagate" the truth value resulting from the evaluation of the body to the head, by taking into account the certainty α_r associated to the rule r, while the disjunction function (*e.g.* $\oplus$) dictates how to combine the certainties in case an atom appears in the heads of several rules.

We further define the *Herbrand base* $\mathcal{HB}_P$ of an np-program P as the set of all instantiated atoms corresponding to atoms appearing in P and define P^* to be the *Herbrand instantiation of* P, *i.e.* the set of all ground instantiations of the rules in P. We can note that any classical logic program P is equivalent to the np-program constructed by replacing each classical rule in P of the form

$$A \leftarrow B_1, ..., B_n, \neg C_1, ..., \neg C_m$$

by the rule

$$r : A \overset{t}{\leftarrow} B_1, ..., B_n, \neg C_1, ..., \neg C_m; \langle \oplus, \otimes, \otimes \rangle$$

where $\mathcal{T} = \{f, t\}$, $\preceq$ is defined by $f \preceq t$, $\oplus = max_{\preceq}$, $\otimes = min_{\preceq}$, $\neg f = t$ and $\neg t = f$.

Example 1. The following example describes a legal case where a judge has to decide whether to charge a person named John accused of murder. To do so, the judge collects facts that he combines using an np-program in order to reach a decision.

Consider the complete lattice $\langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$, where $\mathcal{T}$ is $[0, 1]$, $\forall, a, b \in [0, 1]$, $a \preceq b$ iff $a \leq b$, $a \otimes b = \min(a, b)$, and $a \oplus b = \max(a, b)$. Consider the disjunction function $f_d(\alpha, \beta) = \alpha + \beta - \alpha \cdot \beta$, the conjunction function $f_c(\alpha, \beta) = \alpha \cdot \beta$ and the propagation function $f_p = f_c$. The negation function is the usual function $\neg(\alpha) = 1 - \alpha$. Then the following is an np-program P:[2]

$$P = \left\{ \begin{array}{ll}
\texttt{suspect(X)} & \overset{0.6}{\leftarrow} \texttt{motive(X)} \;\; \langle f_d, \otimes, - \rangle \\
\texttt{suspect(X)} & \overset{0.8}{\leftarrow} \texttt{witness(X)} \;\; \langle f_d, \otimes, - \rangle \\
\texttt{innocent(X)} & \overset{1}{\leftarrow} \texttt{alibi(X,Y)} \wedge \neg \texttt{friends(X,Y)} \;\; \langle f_d, f_p, \otimes \rangle \\
\texttt{friends(X,Y)} & \overset{1}{\leftarrow} \texttt{friends(Y,X)} \;\; \langle \oplus, f_p, - \rangle \\
\texttt{friends(X,Y)} & \overset{0.7}{\leftarrow} \texttt{friends(X,Z)} \wedge \texttt{friends(Z,Y)} \;\; \langle \oplus, f_p, f_c \rangle \\
\texttt{charge(X)} & \overset{1}{\leftarrow} \texttt{suspect(X)}, \neg \texttt{innocent(X)} \;\; \langle \oplus, f_p, - \rangle \\
\texttt{motive(John)} & \overset{1}{\leftarrow} 0.8 \;\; \langle \oplus, f_p, - \rangle \\
\texttt{alibi(John,Sam)} & \overset{1}{\leftarrow} 1 \;\; \langle \oplus, f_p, - \rangle \\
\texttt{friends(John,Ted)} & \overset{1}{\leftarrow} 0.8 \;\; \langle \oplus, f_p, - \rangle \\
\texttt{friends(Sam,Ted)} & \overset{1}{\leftarrow} 0.6 \;\; \langle \oplus, f_p, - \rangle
\end{array} \right\}$$

Some comments on the rules. The two first rules of R describe how a person X is shown to be a suspect, *i.e.* by providing a motive (first rule) or a witness

[2] The symbol $-$ instead of a function denotes the facts that this function is not relevant. Note that any conjunction function is also a propagation function.

against X (second rule). The third rule of R describes how a person X is shown to be innocent, *i.e.* by providing an alibi for X by a person who is not a friend of X. The fourth and fifth rules describe the relation `friend`. Finally, the sixth rule of P is the "decision making rule" and the last rules are the facts collected by the judge. Note that *e.g.* for predicate `suspect`, the disjunction function f_d is associated, as if there are different ways to infer that someone is suspect, then we would like to increase (summing up) our suspicion and not just to choose the maximal value. In the fifth rule, the function f_p allows us to infer some friendship relations taking into account the fact that friendship decreases with transitivity. Moreover, the rules are associated to different propagation coefficients and functions corresponding to the reliability we associate to the information inferred from those rules. □

3 Interpretations of programs

An interpretation of an np-program P is a function that assigns to all atoms of the Herbrand base of P a value in $\mathcal{T}$. We denote $\mathcal{V}_P(\mathcal{T})$ the set of all interpretations of P.

An important issue is to determine which is the intended meaning or semantics of an np-program. Following the usual approach, the semantics of a program P is determined by selecting a particular interpretation of P in the set of models of P. In logic programs without negation, as well as in the parametric IB framework, that chosen model is usually the least model of P w.r.t. $\preceq$.

Introducing negation in classical logic programs, some np-programs do not have a unique minimal model, as shown in the following examples.

Example 2. Let P be the classical program defined by the two rules

$$A \leftarrow \neg B$$
$$B \leftarrow \neg A$$

The program P has two minimal models: $I_1 = \{A\!: f, B\!: t\}$ and $I_2 = \{A\!: t, B\!: f\}$ that are not comparable w.r.t. the truth ordering. □

Of course, we can observe the same problem in our parametric IB framework as shown in the following example.

Example 3. Let $\mathcal{T}$ be $[0, 1]$. Consider, as usual, $f_c(\alpha, \beta) = \min(\alpha, \beta)$, $f_d(\alpha, \beta) = \max(\alpha, \beta)$, $f_p(\alpha, \beta) = \alpha{\cdot}\beta$ and the usual negation function. Consider the program P defined by the following rules :

$$A \overset{1}{\leftarrow} \neg B; \langle f_d, f_p, - \rangle$$
$$B \overset{1}{\leftarrow} \neg A; \langle f_d, f_p, - \rangle$$
$$A \overset{1}{\leftarrow} 0.2; \langle f_d, f_p, - \rangle$$
$$B \overset{1}{\leftarrow} 0.3; \langle f_d, f_p, - \rangle$$

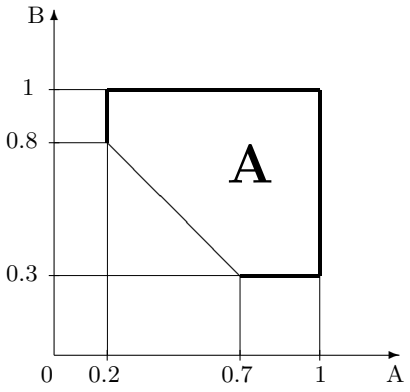

Fig. 1. Infinite minimal models.

This program will have an infinite number of models I_x^y, where $0.2 \leq x \leq 1$, $0.3 \leq y \leq 1$, $y \geq 1 - x$, $I_x^y(A) = x$ and $I_x^y(B) = y$ (those in the A area in Figure 1). There are also an infinite number of minimal models (those on the thin diagonal line) according to the order $(a, b) \preceq (c, d)$ iff $a \preceq c$ and $b \preceq d$. These minimal models I_x^y are such that $y = 1 - x$. □

Concerning the previous example we may note that the truth value of A in the minimal models is in the interval $[0.2, 0.7]$, while for B the interval is $[0.3, 0.8]$. An obvious question is: what should the response to a query A to the program in Example 3 be? There are at least two answers:

1. the truth value of A is *undefined*, as there is no unique minimal model. This is clearly a conservative approach, which in case of ambiguity prefers to leave A unspecified;
2. the truth value of A is in $[0.2, 0.7]$, which means that even if there is no unique value for A, in all minimal models the truth of A is in $[0.2, 0.7]$. In this approach we still try to provide some information. Of course, some care should be used. Indeed from $I(A) \in [0.2, 0.7]$ and $I(B) \in [0.3, 0.8]$ we should not conclude that $I(A) = 0.2$ and $I(B) = 0.3$ is a model of the program.

In this paper we address solution 1. and leave solution 2. for future work. In order to allow some atom's truth value to be unspecified, we will introduce *partial interpretations*: partial interpretations correspond to interpretations that assign values only to some atoms of $\mathcal{HB}_P$ and are not defined for the other atoms.

Definition 2 (Partial interpretation). *Let P be an np-program.* A partial interpretation I *of P is a partial function from $\mathcal{HB}_\mathcal{P}$ to $\mathcal{T}$.* ∎

A partial interpretation I can be seen as a set $\{A : \mu \mid A \in \mathcal{HB}_P \text{ and } \mu \in \mathcal{T}\}$, such that each atom in $\mathcal{HB}_P$ appears at most once in that set, defined by: for all ground atoms A, $A : \mu \in I$ if $I(A) = \mu$. Of course, an interpretation is a

partial interpretation. Interpretations and partial interpretations will be used as functions or as sets following the context.

In the following, given an np-program P, given an interpretation I such that each premise in the body is defined under I, (i) with r_A we denote a rule $(r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n, \neg C_1, ..., \neg C_m; \langle f_d, f_p, f_c \rangle) \in P^*$, whose head is A; and (ii) with $I(r_A)$ we denote the evaluation of the body of r_A w.r.t. I, *i.e.*

$$I(r_A) = f_p(\alpha_r, f_c(\{I(B_1), \ldots, I(B_n), \neg I(C_1), \ldots, \neg I(C_m)\}))$$

$I(r_A)$ is undefined in case some premise in the body is undefined in I, except for the case where there is an i such that $I(B_i) = \bot$ or $I(C_i) = \top$. In that case, we define $I(r_A) = \bot$.

Definition 3 (Satisfaction of an np-program). *Let P be an np-program and let I be a partial interpretation of P. Then we say that I* satisfies *(is a* model *of) P, denoted $\models_I P$, iff $\forall A \in \mathcal{HB}_P$:*

1. *if there is a rule $r_A \in P^*$ such that $I(r_A) = \top$, then $I(A) = \top$;*
2. *if $I(r_A)$ is defined for all rules $r_A \in P^*$, then $I(A) \succeq f_d(\{I(r_A) | r_A \in P^*\})$, where f_d is the disjunction function associated with $\pi(A)$, the predicate symbol of A.* ∎

Example 4. For the program P in Example 2, it is easily verified that the interpretations $I_1 = \{A{:}\, f, B{:}\, t\}$, $I_2 = \{A{:}\, t, B{:}\, f\}$ and $I_3 = \emptyset$ are models of P.

For the program P in Example 3, the interpretations I_x^y such that $0.2 \leq x \leq 1$, $0.3 \leq y \leq 1$, $y \geq 1 - x$, $I_x^y(A) = x$ and $I_x^y(A) = y$ are all models of P. Note that the interpretation $I_4 = \emptyset$ is also a model of P. □

It is worth noting that if we restrict our attention to positive programs only then the definition reduces to that presented in [20] if the interpretation I is not partial but defined for all atoms in $\mathcal{HB}_\mathcal{P}$.

4 Alternating fixpoint and compromise semantics

In this section we will define our well-founded semantics for np-programs.

4.1 Immediate Consequence Operators

First, we extend the ordering on $\mathcal{T}$ to the space of interpretations $\mathcal{V}_P(\mathcal{T})$. Let I_1 and I_2 be in $\mathcal{V}_P(\mathcal{T})$, then $I_1 \preceq I_2$ if and only if $I_1(A) \preceq I_2(A)$ for all ground atoms A. Under this ordering $\mathcal{V}_P(\mathcal{T})$ becomes a complete lattice, and we have $(I_1 \otimes I_2)(A) = I_1(A) \otimes I_2(A)$, and similarly for the other operators. The actions of functions can be extended from atoms to formulas as follows: $I(f_c(X, Y)) = f_c(I(X), I(Y))$, and similarly for the other functions. Finally, for all α in $\mathcal{T}$ and for all I in $\mathcal{V}_P(\mathcal{T})$, $I(\alpha) = \alpha$.

We now define a new operator $T_\mathcal{P}$ inspired by [8, 30–32]. It infers new information from two interpretations: the first one is used to evaluate the positive literals, while the second one is used to evaluate the negative literals of the bodies of rules in P.

Definition 4. *Let P be any np-program. The* immediate consequence operator *T_P is a mapping from $\mathcal{V}_P(\mathcal{T}) \times \mathcal{V}_P(\mathcal{T})$ to $\mathcal{V}_P(\mathcal{T})$, defined as follows: for every pair (I, J) of interpretations in $\mathcal{V}_P(\mathcal{T})$, for every atom A, $T_P(I, J)(A) = f_d(X)$, where f_d is the disjunction function associated with $\pi(A)$, the predicate symbol of A, and*

$$X = \{f_p(\alpha_r, f_c(\{I(B_1), \ldots, I(B_n), \neg J(C_1), ..., \neg J(C_m)\})) : (r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n, \neg C_1, ..., \neg C_m; \langle f_d, f_p, f_c\rangle) \in P^*\}$$

∎

Note that the T_P operator applies to interpretations only and not to partial interpretations. Additionally, the interpretation $T_P(I, J)$ is in $\mathcal{V}_P(\mathcal{T})$.

It is easy to prove that

Proposition 1. *Let P be any np-program. $T_{\mathcal{P}}$ is monotonic in its first argument, and anti-monotone in its second argument w.r.t. $\preceq$.* ∎

4.2 Compromise semantics of an np-program

Using Proposition 1 and the Knaster-Tarski theorem, we can define an operator S_P, inspired from [11] and derived from T_P, that takes an interpretation J as input, first evaluates the negative literals of the program w.r.t. J, and then returns the minimal model of the resulting "positive" np-program w.r.t. $\preceq$.

Let $I_\perp$ be the interpretation that assigns the value $\perp$ to all atoms of $\mathcal{HB}_{\mathcal{P}}$, *i.e.* the minimal element of $\mathcal{V}_P(\mathcal{T})$ w.r.t. $\preceq$, while let $I_\top$ be the interpretation that assigns the value $\top$ to all atoms of $\mathcal{HB}_{\mathcal{P}}$, *i.e.* the maximal element of $\mathcal{V}_P(\mathcal{T})$ w.r.t. $\preceq$.

Definition 5. *Let P be any np-program. We define $S_P(J) = T_P^\infty(I_\perp, J)$,* i.e. *the least fixpoint of T_P w.r.t. $\preceq$ for a given interpretation J.* ∎

Intuitively, $S_P(J)$ is the interpretation that assigns to negative literals the evaluation of them in the fixed interpretation J and then applies the usual T_P operator. Note that the closure ordinal is at most the first ordinal limit ω.

Example 5. Let P be the np-program of Example 1, then we have[3]

$$S_P(I_\perp) = \{\ \texttt{friends(John, Ted)} : 0.8,\quad \texttt{motive(John)} : 0.8,$$
$$\texttt{friends(Ted, John)} : 0.8,\quad \texttt{alibi(John, Sam)} : 1,$$
$$\texttt{friends(Sam, Ted)} : 0.6,\quad \texttt{suspect(John)} : 0.6,$$
$$\texttt{friends(Ted, Sam)} : 0.6,\quad \texttt{innocent(John)} : 1,$$
$$\texttt{friends(John, Sam)} : 0.336,\quad \texttt{charge(John)} : 0.6,$$
$$\texttt{friends(Sam, John)} : 0.336\ \}$$

□

[3] $S_P(I_\perp)$ is a total interpretation, but we will indicate only the atoms whose values are different from 0.

From Proposition 1, we easily derive the following property of S_P.

Proposition 2. *Let P be any np-program, then S_P is anti-monotone w.r.t. $\preceq$ and, thus, $S_P \circ S_P$ is monotone.* ■

There is a well-know property, which derives from the Knaster-Tarski theorem and deals with anti-monotone functions on complete lattices:

Proposition 3 ([40]). *Suppose that a function f is anti-monotone on a complete lattice $\mathcal{L} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$. Then there are two unique elements μ and ν of $\mathcal{T}$, called extreme* oscillation points *of f, such that the following hold:*

- *μ and ν are the least and greatest fixpoint of $f \circ f$ (*i.e. *of f composed with f;*
- *f oscillates between μ and ν in the sense that $f(\mu) = \nu$ and $f(\nu) = \mu$;*
- *if x and y are also elements of $\mathcal{T}$ between which f oscillates then x and y lie between μ and ν.* ■

Under the ordering $\preceq$, S_P is anti-monotone and $\mathcal{V}_P(\mathcal{T})$ is a complete lattice, so S_P has two extreme oscillation points under this ordering.

Proposition 4. *Let P be any np-program, S_P has two extreme oscillation points under $\preceq$, $S_P^{\bot} = (S_P \circ S_P)^{\infty}(I_{\bot})$ and $S_P^{\top} = (S_P \circ S_P)^{\infty}(I_{\top})$, with $S_P^{\bot} \preceq S_P^{\top}$.* ■

As in Van Gelder's alternated fixpoint approach [11], $S_P^{\bot}$ and $S_P^{\top}$ are respectively an under-estimation and an over-estimation of P. As the meaning of P, we propose to consider the consensus or compromise between those two interpretations, *i.e.* to consider as defined only the atoms whose values coincide in both limit interpretations. We define the *compromise* between $S_P^{\bot}$ and $S_P^{\top}$ to be the intersection between $S_P^{\bot}$ and $S_P^{\top}$.

Definition 6. *Let P be any np-program, the* compromise semantics *of P, denoted $CS(P)$, is defined by $CS(P) = S_P^{\bot} \cap S_P^{\top}$.* ∎

The following theorem asserts that, for any np-program P, $CS(P)$ satisfies P and will be seen as its the intended meaning or semantics.

Theorem 1. *Let P be any np-program, then $\models_{CS(P)} P$.*

Sketch of proof. Given an np-program P, it can be proved that for all rules $r_A \in P^*$, if $CS(P)(r_A)$ is defined, then it means that either there is one literal L in the body of r_A such that $CS(P)(L) = \bot$, or all the literals are defined in $CS(P)$. In the first case, we have $S_P^{\bot}(L) = S_P^{\top}(L) = \bot$, and it follows that $S_P^{\bot}(r_A) = S_P^{\top}(r_A) = \bot$. In the second case, for all literals $L \in r_A$, $S_P^{\bot}(L) = S_P^{\top}(L)$, so $S_P^{\bot}(r_A) = S_P^{\top}(r_A)$. It follows that for all atoms $A \in \mathcal{HB}_{\mathcal{P}}$,

- if there is a rule $r_A \in P^*$ such that $CS(P)(r_A) = \top$, then it means that $S_P^{\bot}(A) = S_P^{\top}(A) = \top$, thus $CS(P)(A) = \top$;

– if $CS(P)(r_A)$ is defined for all rules $r_A \in P^*$, then $S_P^{\bot}(r_A) = S_P^{\top}(r_A)$ for all rules $r_A \in P^*$, and we have

$$\begin{aligned} f_d(\{CS(P)(r_A) | r_A \in P^*\}) &= f_d(\{S_P^{\bot}(r_A) | r_A \in P^*\}) \\ &= f_d(\{S_P^{\top}(r_A) | r_A \in P^*\}) \\ &= CS(P)(A) \end{aligned}$$

We can conclude that $CS(P)$ satisfies P. ■

Example 6. Let P be the np-program of Example 1, then we have[4]

$$\begin{aligned} CS(P) = \{ & \texttt{friends}(\texttt{John}, \texttt{Ted}) : 0.8, && \texttt{motive}(\texttt{John}) : 0.8, \\ & \texttt{friends}(\texttt{Ted}, \texttt{John}) : 0.8, && \texttt{alibi}(\texttt{John}, \texttt{Sam}) : 1, \\ & \texttt{friends}(\texttt{Sam}, \texttt{Ted}) : 0.6, && \texttt{suspect}(\texttt{John}) : 0.6, \\ & \texttt{friends}(\texttt{Ted}, \texttt{Sam}) : 0.6, && \texttt{innocent}(\texttt{John}) : 0.664, \\ & \texttt{friends}(\texttt{John}, \texttt{Sam}) : 0.336, && \texttt{charge}(\texttt{John}) : 0.336, \\ & \texttt{friends}(\texttt{Sam}, \texttt{John}) : 0.336 && \} \end{aligned}$$

□

Example 7. For the program P in Example 2, $CS(P) = \emptyset$, and for the program P in Example 3, $CS(P) = \emptyset$. □

Example 8. Consider the following np-program P on $\mathcal{T} = [0, 1]$, with the disjunction function $f_d(\alpha, \beta) = \max(\alpha, \beta)$, the conjunction function $f_c(\alpha, \beta) = \min(\alpha, \beta)$ and the propagation function $f_p(\alpha, \beta) = \alpha \cdot \beta$. Negation is as usual. All rules have the same functions associated, so we omit them for readability.

$$\begin{aligned} A &\overset{0.8}{\leftarrow} B, \neg C \\ B &\overset{0.9}{\leftarrow} A, D \\ D &\overset{0.6}{\leftarrow} 0.2 \end{aligned}$$

It can be shown that $CS(P) = I$, where $I(A) = I(B) = I(C) = 0$ and $I(D) = 0.12$. □

4.3 Comparison with existing semantics

We conclude this section by showing that our extension is a conservative one.

Our semantics extends the Lakshmanan and Shiri's semantics of parametric programs presented in [20] to normal parametric programs. This is due to the fact that the machinery developed in order to deal with negation has no effect in positive programs.

[4] Note that it follows from $S_P^{\bot} = S_P^{\top}$ that $CS(P)$ is a totally defined interpretation. But, for ease of presentation, we will indicate only the atoms whose values are different from 0.

Proposition 5. *If P is an np-program without negation then the compromise semantics $CS(P)$ of P coincides with the Lakshmanan and Shiri's semantics of P.* ■

Example 9. Consider the following np-program P on $\mathcal{T} = [0, 1]$, with the disjunction function $f_d(\alpha, \beta) = \alpha + \beta - \alpha \cdot \beta$, the conjunction function $f_c(\alpha, \beta) = \alpha \cdot \beta$ and the propagation function $f_p = f_c$. Negation is as usual.

$$A \overset{0.6}{\leftarrow} B, C$$
$$A \overset{1}{\leftarrow} D$$
$$B \overset{0.5}{\leftarrow} 0.4$$
$$D \overset{0.3}{\leftarrow} 0.2$$
$$C \overset{1}{\leftarrow} 0.5$$

It is easily to be verified that $CS(P) = I$, where $I(A) = 0.1164$, $I(B) = 0.2$, $I(C) = 0.5$ and $I(D) = 0.06$ □

Finally, we compare our semantics with the well-founded semantics of classical logic programs defined in [28].

Proposition 6. *Let P be a Datalog program with negation. The compromise semantics $CS(P)$ of P coincides with the well-founded semantics of P.* ■

Example 10. Consider the Boolean lattice and the classical logic program P defined by :

$$A \leftarrow B, \neg C$$
$$B \leftarrow A, D$$
$$D \leftarrow$$

Then $CS(P) = I$, where $I(A) = I(B) = I(C) = f$ and $I(D) = t$. This corresponds exactly to the well-founded semantics of the program P. □

5 Conclusions

We have extended the parametric IB approach [20], a general framework for the representation and the manipulation of uncertainty in logic databases and expert systems, with default negation. The semantical approach that we adopted for default negation in logic programs is based on the well-founded semantics, a widely studied and used semantics of classical logic programs with negation. Technically, we integrated the fixpoint semantics of the parametric IB approach with the alternating fixpoint techniques proposed in [11]. We have also shown that our extension is a conservative extension.

In the future we will address the issue in which intervals are specified for those atoms for which no consensus exists.

References

1. Fahiem Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. The MIT Press, 1990.
2. True H. Cao. Annotated fuzzy logic programs. *Fuzzy Sets and Systems*, 113(2):277–298, 2000.
3. K.L. Clark. On closed world data bases. In Hervé Gallaire and Jack Minker, editors, *Logic and data bases*, pages 293–322. Plenum Press, New York, NY, 1978.
4. Alex Dekhtyar and V.S. Subrahmanian. Hybrid probabilistic programs. In *Proc. of the 13th Int. Conf. on Logic Programming (ICLP-97)*, Leuven, Belgium, 1997. The MIT Press.
5. Didier Dubois, Jérome Lang, and Henri Prade. Towards possibilistic logic programming. In *Proc. of the 8th Int. Conf. on Logic Programming (ICLP-91)*, pages 581–595. The MIT Press, 1991.
6. Didier Dubois and Henri Prade. Approximate and commonsense reasoning: From theory to practice. In Zbigniew W. Ras and Michalewicz Maciek, editors, *Proc. of the 9th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-96)*, number 1079 in Lecture Notes in Artificial Intelligence, pages 19–33. Springer-Verlag, 1996.
7. Gonzalo Escalada-Imaz and Felip Manyà. Efficient interpretation of propositional multiple-valued logic programs. In *Proc. of the 5th Int. Conf. on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-94)*, number 945 in Lecture Notes in Computer Science, pages 428–439. Springer-Verlag, 1994.
8. M. C. Fitting. The family of stable models. *Journal of Logic Programming*, 17:197–225, 1993.
9. Melvin Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
10. Norbert Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.
11. Allen Van Gelder. The alternating fixpoint of logic programs with negation. In *Proc. of the 8th ACM SIGACT SIGMOD Sym. on Principles of Database Systems (PODS-89)*, pages 1–10, 1989.
12. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 579–597, Jerusalem, 1990. The MIT Press.
13. Michael Gelfond, Halina Przymusinska, and Teodor C. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:75–94, 1989.
14. Mitsuru Ishizuka and Naoki Kanai. Prolog-ELF: incorporating fuzzy logic. In *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI-85)*, pages 701–703, Los Angeles, CA, 1985.
15. M. Kifer and Ai Li. On the semantics of rule-based expert systems with uncertainty. In *Proc. of the Int. Conf. on Database Theory (ICDT-88)*, number 326 in Lecture Notes in Computer Science, pages 102–117. Springer-Verlag, 1988.
16. Michael Kifer and V.S. Subrahmanian. Theory of generalized annotaded logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
17. R. Kruse, E. Schwecke, and J. Heinsohn. *Uncertainty and Vagueness in Knowledge Based Systems*. Springer-Verlag, Berlin, Germany, 1991.

18. Laks Lakshmanan. An epistemic foundation for logic programming with uncertainty. In *Foundations of Software Technology and Theoretical Computer Science*, number 880 in Lecture Notes in Computer Science, pages 89–100. Springer-Verlag, 1994.
19. Laks V.S. Lakshmanan and Nematollaah Shiri. Probabilistic deductive databases. In *Int'l Logic Programming Symposium*, pages 254–268, 1994.
20. Laks V.S. Lakshmanan and Nematollaah Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
21. James J. Lu. Logic programming with signs and annotations. *Journal of Logic and Computation*, 6(6):755–778, 1996.
22. Thomas Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, pages 388–392, Brighton (England), August 1998.
23. John McCarthy. Circumscription - a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
24. Jack Minker. On indefinite data bases and the closed world assumption. In Springer-Verlag, editor, *Proc. of the 6th Conf. on Automated Deduction (CADE-82)*, number 138 in Lecture Notes in Computer Science, 1982.
25. Raymond Ng and V.S. Subrahmanian. Stable model semantics for probabilistic deductive databases. In Zbigniew W. Ras and Maria Zemenkova, editors, *Proc. of the 6th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-91)*, number 542 in Lecture Notes in Artificial Intelligence, pages 163–171. Springer-Verlag, 1991.
26. Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1993.
27. Raymond Ng and V.S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning*, 10(3):191–235, 1993.
28. Allen nva Gelder, Kenneth A. Ross, and John S. Schlimpf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, January 1991.
29. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Los Altos, 1988.
30. T. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.
31. T. C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.
32. T. C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In S. Debray and H. Hermenegildo, editors, *Logic Programming, Proceedings of the 1990 North American Conference*, pages 40–59. MIT Press, 1990.
33. Teodor C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990.
34. Raymond Reiter. On closed world data bases. In Hervé Gallaire and Jack Minker, editors, *Logic and data bases*, pages 55–76. Plenum Press, New York, NY, 1978.
35. Ehud Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proc. of the 8th Int. Joint Conf. on Artificial Intelligence (IJCAI-83)*, pages 529–532, 1983.
36. V.S. Subramanian. On the semantics of quantitative logic programs. In *Proc. 4th IEEE Symp. on Logic Programming*, pages 173–182. Computer Society Press, 1987.

37. M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Philosophical Logic*, (1):37–53, 1986.
38. Gerd Wagner. Negation in fuzzy and possibilistic logic programs. In T. Martin and F. Arcelli, editors, *Logic programming and Soft Computing*, pages –. Research Studies Press, 1998.
39. Beat Wüttrich. Probabilistic knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):691–698, 1995.
40. S. Yablo. Truth and reflection. *Journal of Philosophical Logic*, 14:297–349, 1985.