Uncertainty and Partial Non-Uniform Assumptions in Parametric Deductive Databases

Yann Loyer and Umberto Straccia

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

Abstract. Different many-valued logic programming frameworks have been proposed to manage uncertain information in deductive databases and logic programming. A feature of these frameworks is that they rely on a predefined assumption or hypothesis, i.e. an interpretation that assigns the same default truth value to all the atoms of a program, e.g. in the open world assumption, by default all atoms have unknown truth value. In this paper we extend these frameworks along three directions: (*i*) we will introduce non-monotonic modes of negation; (*ii*) the default truth values of atoms need not necessarily to be all equal each other; and (*iii*) a hypothesis can be a partial interpretation. We will show that our approach extends the usual ones: if we restrict our attention to classical logic programs and consider total uniform hypotheses, then our semantics reduces to the usual semantics of logic programs. In particular, under the everything false assumption, our semantics captures and extends the well-founded semantics to these frameworks.

1 Introduction

An important issue to be addressed in applications of logic programming is the management of uncertainty whenever the information to be represented is of imperfect nature (which happens quite often). The problem of uncertainty management in logic programs has attracted the attention of many researchers and numerous frameworks have been proposed [1–3, 6, 8–11, 13–17]. Each of them addresses the management of different kind of uncertainty: (*i*) probability theory [6, 10, 13–15]; (*ii*) fuzzy set theory [1, 16, 17]; (*iii*) multi-valued logic [8, 9, 11]; and (*iv*) possibilistic logic [3]. Apart from the different notion of uncertainty they rely on, these frameworks differ in the way in which uncertainty is associated with the facts and rules of a program. With respect to this latter point, these frameworks can be classified into *annotation based* (AB) and *implication based* (IB), which we briefly summarize below. In the AB approach, 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 atom A is at least (or is in) $f(\beta_1, \ldots, \beta_n)$, whenever the certainty of atom B_i is at least (or is in) β_i, $1 \leq i \leq n$". Here f is an n-ary computable function and β_i is either a constant or a variable ranging over an appropriate certainty domain. Examples of AB frameworks include [8, 9, 14, 15]. In the IB approach, a rule is of the form $A \overset{\alpha}{\leftarrow} B_1, ..., B_n$, which says that the certainty associated with the implication $B_1 \wedge ... \wedge B_n \rightarrow A$ is α. Computationally, given an assignment v of certainties to the B_is, the certainty of A is computed by taking the "conjunction" of the certainties $v(B_i)$ and then somehow "propagating" it to the rule head. The truth values are taken from a certainty lattice. Examples of the IB frameworks include [10, 11, 17] (see [11] for a more detailed comparison between the two approaches). We recall the following facts [11]: (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. Nonetheless both approaches stress important limitations for real-world applications, which we will address in this paper: (i) they do not address any mode of *non-monotonic reasoning* (in particular, no negation operation is defined). The need of non-monotonic formalisms for real-world applications is commonly accepted: our knowledge about the world is almost always *incomplete* and, thus, we are forced to reason in the *absence of complete information.* As a result we often have to revise our conclusions, when new information becomes available; (ii) they rely on predefined *uniform assumptions*, *i.e.* they assign the same default truth value to all the atoms. Widely used and well known examples of uniform assumptions are the *Open World Assumption* (OWA), which corresponds to the assumption that every atom whose truth can not be inferred from the program has *unknown* truth value, and the *Closed World Assumption* (CWA), which corresponds to the assumption that every such atom's truth value is *false.* While only uniform assumptions are used, the need of non-uniform assumptions has already been highlighted in the domains of (a) logic-based information retrieval [6], and (b) information integration [12], as shown in the following example.

Example 1 (A motivating example). Consider a legal case where a judge has to decide whether to charge a person named Ted accused of murder. To do so, the judge first collects facts from two different sources: the public prosecutor and the person's lawyer. The judge then combines the collected facts using a set of rules in order to reach a decision. For the sake of our example, let us suppose that the judge has collected a set of facts $F = \{\mathtt{witness(John)}, \mathtt{friends(John, Ted)}\}$ that he combines using a set of rules R as follows[1]:

$$R = \left\{ \begin{array}{ll} \mathtt{suspect(X)} & \leftarrow \mathtt{motive(X)} \\ \mathtt{suspect(X)} & \leftarrow \mathtt{witness(X)} \\ \mathtt{innocent(X)} & \leftarrow \mathtt{alibi(X,Y)} \wedge \neg\mathtt{friends(X,Y)} \\ \mathtt{innocent(X)} & \leftarrow \mathtt{presumption\_of\_innocence(X)} \wedge \neg\mathtt{suspect(X)} \\ \mathtt{friends(X,Y)} & \leftarrow \mathtt{friends(Y,X)} \\ \mathtt{friends(X,Y)} & \leftarrow \mathtt{friends(X,Z)} \wedge \mathtt{friends(Z,Y)} \\ \mathtt{charge(X)} & \leftarrow \mathtt{suspect(X)} \\ \mathtt{charge(X)} & \leftarrow \neg\mathtt{innocent(X)} \end{array} \right\}$$

The two first rules of R describe how the prosecutor works: in order to support the claim that a person X is a suspect, the prosecutor tries to provide a motive (first rule) or a witness against X (second rule). The third and fourth rules of R describe how the lawyer works: in order to support the claim that X is innocent,

[1] For ease of presentation we leave uncertainties out. For instance, there might well be some uncertainty in the facts and rules, as we will see later on.

the lawyer tries to provide an alibi for X by a person who is not a friend of X (third rule), or to use the presumption of innocence if the person is not suspect (fourth rule), *i.e.* the defendant is *assumed* innocent until proved guilty. Finally, the last two rules of R are the judge's "decision making rules".

Now, what should the value of `charge(Ted)` be? `Ted` should be charged if he has been proved effectively suspect or not innocent. We can easily remark that uniform hypotheses, as the CWA and the OWA, do not fit with our expectation. According to the CWA, then the judge will infer that Ted is not innocent and must be charged. According to the OWA, then the atoms `suspect(Ted)`, `innocent(Ted)` and `charged(Ted)` are unknown and the judge cannot take a decision (that semantics is often considered as too weak). Another uniform hypothesis is to assign to all atoms the default value true, but under such an assumption, the judge will infer that Ted is suspect and must be charged.

An intuitively appealing *non-uniform assumption* in this situation is to assume by default that the atoms `motive(Ted)`, `witness(Ted)` and `suspect(Ted)` are false, that the atom `presumption_of_innocence(Ted)` is true and that the others are unknown. With such a hypothesis, the judge could infer that Ted is innocent, not suspect and should not be charged. □

We believe that we should not be limited to consider logic program without negation under uniform assumptions only, but be able to associate to any logic program a semantics based on any given hypothesis, which represents our default or assumed knowledge. To this end, we will extend the *parametric* IB framework [11], a unifying umbrella for IB frameworks, along three directions: (*i*) we will introduce negation into the programs, *i.e.* we will extend the IB frameworks with the non-monotonic mode of negation and extend to that framework the usual semantics of logic programs with negation; (*ii*) we will consider assumptions that do not necessarily assign the same default truth values to all the atoms; and (*iii*) we will not only consider total assumptions but also partial assumptions, *i.e.* we will not require that every atom has a default truth value (an atom's truth value may be still unknown). We will show that our approach extends the usual ones: restricting our attention to logic programs and considering total uniform hypotheses, then our semantics reduces to the usual semantics of logic programs. In particular, under the *everywhere false assumption* (*i*) for programs without negation we obtain the semantics presented in [11]; and (*ii*) for Datalog programs with negation we obtain the *Well Founded Semantics* (WFS) [18]. On the other hand, under the *almost everywhere empty hypothesis* our semantics includes the *Kripke-Kleene semantics* of Fitting [4].

In the next section we introduce the syntax of our logical language, we define the notion of satisfiability and present fixpoint operators, through which in Section 3 the intended semantics of our logic programs is specified. Section 4 compares our semantics with others, while Section 5 concludes.

2 Syntax and semantics: preliminaries

We recall the syntactical aspects of the parametric IB framework presented in [11] and extend it with negation. Let $\mathcal{L}$ be an arbitrary first order language that contains infinitely many variable symbols, finitely many constants, and predicate symbols, but no function symbols. While $\mathcal{L}$ 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 $\langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$ be a certainty lattice (a complete lattice) and $\mathcal{B}(\mathcal{T})$ the set of finite multisets over $\mathcal{T}$. 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. 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. $f(\alpha, \top) = \alpha, \forall \alpha \in \mathcal{T}$.

We require that [11]: (i) any conjunction function in $\mathcal{F}_c$ satisfies properties 1, 2, 3, 5, 6, 7, 9 and 10; (ii) any propagation function in $\mathcal{F}_p$ satisfies properties 1, 2, 3 and 10; and (iii) any disjunction function in $\mathcal{F}_d$ satisfies properties 1, 2, 4, 5, 6, 7 and 8. We also assume that there is a *negation function* $\neg : \mathcal{T} \to \mathcal{T}$ 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 tuple $\langle \mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{P}, \mathcal{D} \rangle$, defined as follows:*

1. *$\mathcal{T}$ is a set of truth values partially ordered by $\preceq$. We assume that $\langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$ is a complete lattice, where $\otimes$ is the meet operator and $\oplus$ the join operator;*
2. *$\mathcal{R}$ is a finite set of* normal parametric rules *r (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. *$\mathcal{C}$ maps each np-rule to a conjunction function in $\mathcal{F}_c$;*
4. *$\mathcal{P}$ maps each np-rule to a propagation function in $\mathcal{F}_p$;*
5. *$\mathcal{D}$ maps each predicate symbol in P to 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 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 is head of several rules. Note that np-programs without negation are parametric programs (*p-programs*) as defined in [11]. We further define the *Herbrand* *base* $\mathcal{HB}_P$ of a 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. A classical logic program is a np-program such that $\otimes$ is the unique conjunction and propagation function, $\oplus$ is the unique disjunction function and $\alpha_r = \top$, for all rules $r \in P$. Such a program will be denoted in the classical way. If not stated otherwise, with P we will always denote an np-program.

Example 2. 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 $f_d(\alpha, \beta) = \alpha + \beta - \alpha \cdot \beta$, $f_c(\alpha, \beta) = \alpha \cdot \beta$, $f_p = f_c$ and $\neg(\alpha) = 1 - \alpha$. Then the following is a np-program P:

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

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 our suspicion and not just to choose the maximal value. □

An interpretation of a 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. Of course, an important issue is to determine which is the semantics of a np-program. In 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, and in particular in our parametric IB framework, has as consequence that some np-programs do not have a unique minimal model.

Example 3. Consider $\mathcal{T} = [0,1]$, $f_c(\alpha,\beta) = \min(\alpha,\beta)$, $f_d(\alpha,\beta) = \max(\alpha,\beta)$, $f_p(\alpha,\beta) = \alpha \cdot \beta$ and the usual negation function. Consider the program $P = \{$ $A \overset{1}{\leftarrow} \neg B; \langle f_d, f_p, f_c \rangle, B \overset{1}{\leftarrow} \neg A; \langle f_d, f_p, f_c \rangle, A \overset{1}{\leftarrow} 0.2; \langle f_d, f_p, f_c \rangle, B \overset{1}{\leftarrow} 0.3; \langle f_d, f_p, f_c \rangle \}$. This program will have an infinite number of models I^y_x, where $0.2 \leq x \leq 1$, $0.3 \leq y \leq 1$, $y \geq 1 - x$, $I^y_x(A) = x$ and $I^y_x(B) = y$. There are also an infinite number of minimal models. These models I^y_x are such that $y = 1 - x$. □

A usual way to deal with such situations in classical logic consists in considering partial interpretations *i.e.* interpretations that assign values only to some atoms of $\mathcal{HB}_P$ and are not defined for the other atoms. A *partial interpretation* I of P is a partial function from $\mathcal{HB}_{\mathcal{P}}$ to $\mathcal{T}$. A partial interpretation I can also be seen as the set $\{A : I(A) \mid A \in \mathcal{HB}_P \text{ and } I(A) \text{ defined}\}$. Partial interpretations will be used as functions or as sets following the context. Furthermore, in the following, given a np-program P, (i) we denote with r_A 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; (ii) given an interpretation I such that each premise in the body of r_A is defined under I, with $I(r_A)$ we denote the evaluation of the body of r_A w.r.t. I, *i.e.* $I(r_A) = f_p(\alpha_r, f_c(\{\!|I(B_1), \ldots, I(B_n), \neg I(C_1), \ldots, \neg I(C_m)|\!\}))$; and (iii) $I(r_A)$ 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 2 (Satisfaction of a np-program). *A partial interpretation 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 for all rules $r_A \in P^*$, $I(r_A)$ is defined, then $I(A) \succeq f_d(X)$, where $X = \{\!|I(r_A) : r_A \in P^*|\!\}$. f_d is the disjunction function associated with $\pi(A)$, the predicate symbol of A.* ∎

Example 4. In Example 3, the interpretations I_x^y are all models of P. The interpretation I undefined on A and B is a model of P as well. □

Restricting our attention to positive programs only, the definition of satisfiability of np-programs reduces to that of satisfiability of p-programs defined in [11], where the interpretation I is not partial but total, *i.e.* defined for all atoms in $\mathcal{HB}(P)$. Note that for total interpretations, case 1 is a consequence of case 2.

From now on, for ease of presentation and without loss of generality, by "programs" we mean "instantiated" programs. 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 formulae 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_P^H inspired by [5, 12]. That operator is parameterized by an interpretation H on $\{\bot, \top\}$. That interpretation represents our default knowledge and we will call it a *hypothesis* to stretch the fact that it represents default knowledge and not "sure knowledge". Such a hypothesis asserts that some atoms are assumed $\bot$ (false) and some others are assumed to be $\top$ (true). In the context of logic programming with uncertainty, a more general approach would consist in considering any interpretation over $\mathcal{T}$ as a possible assumption. Nevertheless, the need of non-uniform assumptions like those of the type defined in this paper, *i.e.* by default the truth value of an atom maybe *true*, *false* or *unknown*, where already considered of interest by the literature [6, 12]. We are aware that this is a limitation and that allowing a hypothesis being any interpretation would give us the most complete approach. But yet unresolved computational difficulties prevent us to consider the generalised case.

The operator T_P^H 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 3 (Parameterized immediate consequence operator). *Let P and H be any np-program and a hypothesis, respectively. The* immediate consequence operator *T_P^H 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, if there is no rule in P with A as its head, then $T_P^H(I, J)(A) = H(A)$, else $T_P^H(I, J)(A) = f_d(X)$, 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 in case negation is absent and H assigns the value $\perp$ to all the atoms, then T_P^H reduces to the immediate consequence operator defined in [11].

Proposition 1. *T_P^H is monotonic in its first argument, and anti-monotonic in its second argument w.r.t. $\preceq$.* ■

Using Proposition 1 and the Knaster-Tarsky theorem, we can define an operator S_P^H, likewise [7] and derived from T_P^H, that takes an interpretation J as input, first evaluates the negative literals of the program w.r.t. J, and then returns the model of the resulting "positive" np-program obtained by iterations of T_P^H beginning with H. But, in order to deal with non-uniform hypotheses, we need first to define the notion of stratification w.r.t. positive cycles.

Definition 4 (Extended positive cycle). *A* positive cycle *of P is a set of rules $\{r_{A_1}, ..., r_{A_n}\}$ of P such that for $1 \leq i \leq n$, A_i (resp. A_n) appears positively in the body of $r_{A_{i+1}}$ (resp. r_{A_1}). An* extended positive cycle *is a positive cycle C extended with the rules in P whose head is the head of one of the rules of C.* ▮

Definition 5 (Stratification w.r.t. extended positive cycles). *A* stratification w.r.t. extended positive cycles *of a np-program P is a sequence of np-programs $P_1, ..., P_n$ such that for the mapping σ from rules of P to [1...n],*

1. *$P_1, ..., P_n$ is a partition of P and every rule r is in $P_{\sigma(r)}$;*
2. *for any r_1 and r_2 in P, where the head of r_2 appears in the body of r_1 and there is no extended positive cycle of P containing r_1, $\sigma(r_1) = \sigma(r_2)$ holds;*
3. *for any r_1 and r_2 in P appearing in the same extended positive cycle, $\sigma(r_1) = \sigma(r_2)$ holds;*
4. *if r_1 and r_2 in P are such that the head of r_2 appears in the body of r_1 and there is an extended positive cycle of P containing r_1 but no extended positive cycle of P containing both r_1 and r_2, then $\sigma(r_2) < \sigma(r_1)$ holds.* ▮

Note that every np-program has a stratification w.r.t. extended positive cycles.

Proposition 2. *Given a np-program P with a stratification $P_1 = P$ w.r.t. extended positive cycles, an interpretation J be over $\mathcal{T}$ and a partial hypothesis H over $\{\perp, \top\}$ such that for all extended positive cycles $\{r_{A_1}, ..., r_{A_k}\}$ of P, $H(A_1) = ... = H(A_k)$ holds. Then the sequence defined by $a_0 = H$ and $a_{n+1} = T_P^H(a_n, J)$ converges.* ■

Definition 6 (The parameterized alternating operator S_P^H). *Consider P with stratification w.r.t. the extended positive cycles $P_1, ..., P_n$ and let J be an interpretation over $\mathcal{T}$. Let H be an interpretation over $\{\bot, \top\}$ such that for all extended positive cycles $\{r_{A_1}, ..., r_{A_k}\}$ of P, $H(A_1) = ... = H(A_k)$ holds. Then $S_P^H(J)$ is the limit of the following sequence: (i) a_1 is the iterated fixpoint of the function $\lambda x.T_{P_1}^H(x, J)$ obtained when beginning the computation with H; (ii) a_i is the iterated fixpoint of the function $\lambda x.T_{P_1\cup...\cup P_i}^H(x, J)$ obtained when beginning the computation with a_{i-1}.* ∎

Intuitively, during the computation of $S_P^H(J)$, we fix the value of the negative premises in P with their values in J. Then we consider the "positive program" and evaluate that program stratum by stratum. After the evaluation of a stratum, we know that the knowledge obtained cannot be modified by what we will infer by activating the rules of the next strata. While a program may have more than one stratification w.r.t. extended positive cycles, the result of the computation does not depend on the stratification used for the computation. Note that the notion of stratification and the condition on H that we have introduced are indispensable for the convergence of the computation.

Example 5. Let $H = \{A : \top, B : \bot, C : \bot, D : \bot\}$ be a hypothesis and let $P = \{A \leftarrow \bot, B \leftarrow \bot, C \leftarrow A, D \leftarrow B, C \leftarrow D, D \leftarrow C\}$. If we compute the sequence $I_0 = H$, $I_i = T_P^H(I_{i-1}, J)$ then we have $I_1 = \{A : \bot, B : \bot, C : \top, D : \bot\}$, $I_2 = \{A : \bot, B : \bot, C : \bot, D : \top\}$, $I_3 = \{A : \bot, B : \bot, C : \top, D : \bot\}, \ldots$ This computation does not terminate. If we consider the definition of S_P^H, then we have a stratification of P with two strata: the first one contains the two first rules of P and the second one the four last rules of P. The computation terminates, and we have $I_1 = I_2 = \{A : \bot, B : \bot, C : \bot, D : \bot\}$. □

Example 6. Let $P = \{A \leftarrow B, B \leftarrow A\}$ and $H = \{A : \top, B : \bot\}$. The condition on H is not satisfied, *i.e.* $H(A) \neq H(B)$, and the computation does not terminate. □

3 Semantics under non-uniform assumptions

In this section we will determine what model, among all the models, is the intended model of a np-program w.r.t. a given hypothesis. For the rest of the paper, if not stated otherwise, *any hypothesis is supposed to assign the same default value to the atoms that are heads of rules of a same extended positive cycle.* From Proposition 1, we derive the following property of S_P^H.

Proposition 3. *Given P and a total non-uniform hypothesis H, S_P^H is anti-monotone w.r.t. $\preceq$ and, thus $S_P^H \circ S_P^H$ 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 4 ([19]). *For a anti-monotone function f on a complete lattice $\mathcal{T}$, there are two* extreme oscillation points *of f, μ and ν in $\mathcal{T}$, such that: (i) μ and ν are the least and greatest fixpoint of the composition $f \circ f$; (ii) f oscillates between μ and ν,* i.e. *$f(\mu) = \nu$ and $f(\nu) = \mu$; (iii) 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^H is anti-monotone and $\mathcal{V}_P(\mathcal{T})$ is a complete lattice, so S_P^H has two extreme oscillation points under this ordering. Let $I_\perp$ be the interpretation that assigns the value $\perp$ to all atoms of $\mathcal{HB}(P)$, 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 $\mathcal{HB}(P)$, i.e. the maximal element of $\mathcal{V}_P(\mathcal{T})$ w.r.t. $\preceq$.

Proposition 5. *Let P and H be any np-program and a total non-uniform hypothesis, respectively. S_P^H has two extreme oscillation points, $S_\perp^H = (S_P^H \circ S_P^H)^\infty(I_\perp)$ and $S_\top^H = (S_P^H \circ S_P^H)^\infty(I_\top)$, with $S_\perp^H \preceq S_\top^H$.*[2] ■

Similarly to van Gelder's alternating fixpoint approach [7], $S_\perp^H$ and $S_\top^H$ are respectively a under-estimation and an over-estimation of P, but w.r.t. any hypothesis H. As the meaning of P we propose to consider as defined only the atoms whose values coincide in both limit interpretations. The truth value of atoms, whose truth value oscillates the "unknown" value is assigned.

Proposition 6. *Let P and H be any np-program and a total non-uniform hypothesis, respectively. Then $\models_{S_\perp^H \cap S_\top^H} P$.* ■

The interpretation $S_\perp^H \cap S_\top^H$ is a model of P, and will be the intended meaning or semantics of P w.r.t. the assumption H.

Definition 7 (Compromise semantics). *The* compromise semantics *of np-program P w.r.t. a total non-uniform assumption H is defined by $M^H(P) = S_\perp^H \cap S_\top^H$.* ▮

Example 7. Let P be the np-program of Example 2 and $I_\perp$, then we have[3] $M^{I_\perp}(P) \supset \{$`s(John)`:0.8, `s(Jim)`:0.6, `s(Ted)`:0, `i(John)`:0, `i(Jim)`:0.664, `i(Ted)`:0, `c(John)`:1, `c(Jim)`:0.6, `c(Ted)`:1$\}$. Now let H be the following hypothesis $H = \{$`m(X)`:0, `w(x)`:0, `s(X)`:0, `p(X)`:1, `a(X,Y)`:0, `f(X,Y)`:0, `i(X)`:1, `c(X)`:0$\}$. Then we have $M^H(P) \supset \{$`s(John)`:0.8, `s(Jim)`:0.6, `s(Ted)`:0, `i(John)`:0.2, `i(Jim)`:0.7984, `i(Ted)`:1, `c(John)`:0.8, `c(Jim)`:0.6, `c(Ted)`:0$\}$. □

Finally, let us briefly show the difficulties introduced in case generalised hypotheses are allowed, as anticipated previously.

Example 8. Given the np-program $P = \{A \leftarrow B, C; B \leftarrow A; B \leftarrow D\}$ and the hypothesis H such that $H(A) = 0.3, H(B) = 0.3, H(C) = 0.5, H(D) = 0.5$. As disjunction and conjunction functions, we will consider the functions $f_d(x,y) = \min\{1, x+y\}$ and $f_c(x,y) = \max\{0, x+y-1\}$, which are also known as the Łukasiewicz disjunction and conjunction, respectively. Consider any interpretation J for negative literals. Now, it can be verified that the iterated fixpoint a_1 of the function $\lambda x.T_P^H(x, J)$ obtained when beginning the computation with H, as specified in Definition 6, does not exist. Indeed, the iteration oscillates between the two interpretations $I = \{A\colon 0, B\colon 0.8, C\colon 0.5, D\colon 0.5\}$ and $I' = \{A\colon 0.3, B\colon 0.5, C\colon 0.5, D\colon 0.5\}$. Therefore, the limit of $S_P^H(J)$ is undefined and, thus, the compromise semantics of P is undefined as well. □

[2] For ease, we omit the P in $S_\perp^H$ and $S_\top^H$.

[3] In the following, for ease of presentation, we will denote each predicate symbol of P by its first letter. Moreover, in the different semantics of P, we will indicate only the values of atoms associated to the symbols of predicates `suspect`, `innocent` and `charge`.

Until know we dealt with total non-uniform assumptions. Let us now address the case where assumptions may be partial. A possible idea to deal with such a situation is to introduce a new logical value u and define a new lattice $\mathcal{T}' = \mathcal{T} \cup \{u\}$. That value represents an unknown or undefined value that means that u is used to replace a value in $\mathcal{T}$ that is currently not known. We would like to extend conjunction and disjunction functions in the following way: (i) $f_c(u, x) =$ if $x \neq \bot$ then u else $\bot$; and (ii) $f_d(u, x) =$ if $x \neq \top$ then u else $\top$. We need also to extend the order $\preceq$ in $\mathcal{T}$ to $\mathcal{T}'$. We know that for all $x \in \mathcal{T}'$, $\bot \preceq x \preceq \top$. But, from the first constraint it follows that $u \preceq x$ for all $x \neq \bot$ and, similarly, from the second one it follows that $x \preceq u$ for all $x \neq \top$. But then, there is a solution only if $\mathcal{T} = \{\bot, \top\}$. We follow another way: a partial hypothesis H on $\{\bot, \top\}$ can be seen as the intersection between two total interpretations $\underline{H}$ and $\overline{H}$, where $\underline{H}$ is as H except that $\bot$ is assumed for the unknown atoms, while $\overline{H}$ is as H except that $\top$ is assumed for the unknown atoms. Note that $\underline{H} \cap \overline{H} = H$. In order to assign a semantics to a np-program w.r.t. such a partial interpretation, we propose to consider the intersection or consensus between the two semantics.

Proposition 7. *Let P be any np-program and let H be a partial non-uniform hypothesis. It follows that $\models_{M^{\underline{H}}(P) \cap M^{\overline{H}}(P)} P$.* ■

Definition 8 (Consensus semantics w.r.t. H). *Let P be any np-program and let H be a partial non-uniform hypothesis. The* consensus semantics *of P w.r.t. H, $\mathcal{C}^H(P)$, is defined by $\mathcal{C}^H(P) = M^{\underline{H}}(P) \cap M^{\overline{H}}(P)$.* [4] ▮

Example 9. Given P of Example 2 and H as suggested in the introduction, *i.e.* $H = \{$`m(X)`:0, `w(x)`:0, `s(X)`:0, `p(X)`:1$\}$, it follows that $\mathcal{C}^H(P) \supset \{$`s(John)` : 0.8, `s(Jim)`:0.6, `s(Ted)`:0, `i(Ted)`:1, `c(John)`:0.8, `c(Jim)`:0.6, `c(Ted)`:0$\}$. □

4 Comparisons with usual semantics

One obvious question would be whether hypothesis maybe simulated by just adding the facts to the program and compute the semantics according to classical approach. This is not true as the following example shows.

Example 10. Let $P = \{A \leftarrow B; B \leftarrow A; D \leftarrow C; C \leftarrow D\}$ and consider the assumption H such that $H(A) = \bot$, and $H(B) = \bot$. The consensus semantics is $\{A{:}\bot, B{:}\bot\}$. Now, suppose that we add $A \leftarrow \bot$ and $B \leftarrow \bot$ to P. Then we can consider two cases, whether we are using the OWA or the CWA. In the former case, the classical semantics of the program will be $\emptyset$, while in the latter example the classical semantics of the program will be $\{A{:}\bot, B{:}\bot, C{:}\bot; D{:}\bot\}$, which are both different from the consensus semantics. □

Our semantics extends the Lakshmanan and Shiri's semantics [11] of parametric programs to normal parametric programs. This is due to the fact that the machinery developed in order to deal with negation has no effect on positive programs, thus for the everywhere false hypothesis $I_\bot$, we have

[4] Note that any compromise semantics is also a consensus semantics.

Proposition 8. *If P is a np-program without negation then the compromise semantics $M^{I_\perp}(P)$ of P (or equivalently the consensus semantics $\mathcal{C}^{I_\perp}(P)$) w.r.t. the hypothesis $I_\perp$ coincides with the Lakshmanan and Shiri's semantics of P.* ■

Now, we compare our semantics with the well-founded semantics [18].

Proposition 9. *Let P be a Datalog program with negation. The compromise semantics $M^{I_\perp}(P)$ of P (or equivalently the consensus semantics $\mathcal{C}^{I_\perp}(P)$) w.r.t. the hypothesis $I_\perp$ coincides with the well-founded semantics of P.* ■

Our approach extends the well-founded semantics to the IB framework as well.

Example 11. Consider Example 2 and hypothesis $I_\perp$. Given the Datalog program with negation P' replacing in P all the truth values by 1, $f_d = \oplus$ and $f_c = \otimes$, $\mathcal{C}^{I_\perp}(P') \supset \{$`s(John)`:1, `s(Jim)`:1, `s(Ted)`:0, `i(John)`:0, `i(Jim)`:0, `i(Ted)`:0, `c(John)`:1, `c(Jim)`:1, `c(Ted)`:1$\}$ follows. □

Finally, we can compare the semantics with the usual semantics and in particular with the Kripke-Kleene semantics [4]. Let us define the *almost everywhere empty hypothesis* H_P w.r.t. P as follows: $H(A) = \perp$ if A is not the head of any rule, else $H(A)$ is undefined.

Proposition 10. *Let P be a Datalog program with negation and H_P the almost everywhere empty hypothesis w.r.t. P. Let $WFS(P)$ be the well-founded semantics of P and $KK(P)$ be the Kripke-Kleene semantics of P. Then $KK(P) \subseteq \mathcal{C}^{H_P}(P) \subseteq WFS(P)$ holds.* ■

The consensus semantics $\mathcal{C}^{H_P}(P)$ of P represents more knowledge than the Kripke-Kleene semantics of P, but less than the well-founded semantics of P. This results shows the well-known fact that the Kripke-Kleene semantics of P is weaker than the well-founded semantics of P.

Example 12. For $P = \{B \leftarrow A, B \leftarrow \neg A, A \leftarrow A\}$, $KK(P) = \emptyset \subset \mathcal{C}^{H_P}(P) = \{B : \top\} \subset WFS(P) = \{A : \perp, B : \top\}$ holds. □

5 Conclusion

We have presented a framework for reasoning about uncertainty and negation in deductive databases and logic programming. Our framework uses parameterized semantics of implication-based logic programming with negation under non-uniform assumptions for the missing information. We have also seen that when we restrict our framework to uniform assumptions only, our approach captures and extends the semantics of conventional logic programs. Obviously, having considered a restricted, still useful, form of hypothesis, puts the generalisation to arbitrary hypothesis to be our primary topic for future work.

References

1. True H. Cao. Annotated fuzzy logic programs. *Fuzzy Sets and Systems*, 113(2):277–298, 2000.
2. Alex Dekhtyar and V.S. Subrahmanian. Hybrid probabilistic programs. *Journal of Logic Programming*, 43(3):187–250, 2000.
3. 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.
4. Melvin Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
5. Melvin Fitting. The family of stable models. *Journal of Logic Programming*, 17(2/3 & 4):197–225, 1993.
6. Norbert Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.
7. 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.
8. 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.
9. Michael Kifer and V.S. Subrahmanian. Theory of generalized annotaded logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
10. Laks V.S. Lakshmanan and Nematollaah Shiri. Probabilistic deductive databases. In *Int'l Logic Programming Symposium*, pages 254–268, 1994.
11. 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.
12. Y. Loyer, N Spyratos, and D. Stamate. Integration of information in four-valued logics under non-uniform assumptions. In *Proceedings of the 30th IEEE International Symposium on Multi-Valued Logics (ISMVL 2000)*, pages 185–191, Portland, Oregon, 2000. IEEE Press.
13. Thomas Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, pages 388–392, Brighton (England), August 1998.
14. 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.
15. Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1993.
16. 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.
17. M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Philosophical Logic*, (1):37–53, 1986.
18. Allen Van 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.
19. S. Yablo. Truth and reflection. *Journal of Philosophical Logic*, 14:297–349, 1985.