The Approximate Well-founded Semantics for Logic Programs with Uncertainty

Yann Loyer[1] and Umberto Straccia[2]

[1] PRiSM, Université de Versailles, 45 Avenue des Etats-Unis, 78035 Versailles, FRANCE
[2] I.S.T.I. - C.N.R., Via G. Moruzzi,1 I-56124 Pisa, ITALY

Abstract. The management of uncertain information in logic programs becomes to be important whenever the real world information to be represented is of imperfect nature and the classical crisp *true, false* approximation is not adequate. A general framework, called *Parametric Deductive Databases with Uncertainty* (PDDU) framework [10], was proposed as a unifying umbrella for many existing approaches towards the manipulation of uncertainty in logic programs. We extend PDDU with (non-monotonic) negation, a well-known and important feature of logic programs. We show that, dealing with uncertain and incomplete knowledge, atoms should be assigned only approximations of uncertainty values, unless some assumption is used to complete the knowledge. We rely on the closed world assumption to infer as much default "false" knowledge as possible. Our approach leads also to a novel characterizations, both epistemic and operational, of the well-founded semantics in PDDU, and preserves the continuity of the immediate consequence operator, a major feature of the classical PDDU framework.

1 Introduction

The management of uncertainty within deduction systems is an important issue whenever the real world information to be represented is of imperfect nature. In logic programming, the problem has attracted the attention of many researchers and numerous frameworks have been proposed. Essentially, they differ in the underlying notion of uncertainty (e.g. probability theory [9, 13–15], fuzzy set theory [16, 17, 19], multi-valued logic [7, 8, 10], possibilistic logic [2]) and how uncertainty values, associated to rules and facts, are managed. Lakshmanan and Shiri have recently proposed a general framework [10], called *Parametric Deductive Databases with Uncertainty* (PDDU), that captures and generalizes many of the precedent approaches. In [10], a rule is of the form $A \overset{\alpha}{\leftarrow} B_1, ..., B_n$. 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, taking into account the certainty α of the implication. However, despite its generality, one fundamental issue that remains unaddressed in PDDU is non-monotonic negation, a well-known and important feature in logic programming.

In this paper, we extend PDDU [10] to *normal logic programs*, logic programs with negation. In order to deal with knowledge that is usually not only uncertain, but also incomplete, we believe that one should rely on approximations of uncertainty values only. Then we study the problem of assigning a semantics to a normal logic program in such a framework. We first consider the least model, show that it extends the Kripke-Kleene semantics [4] from Datalog programs to normal logic programs, but that it is usually to weak. We then explain how one should try to determine approximations as much precise as possible by completing its knowledge by a kind of default reasoning based on the well-known Closed World Assumption (CWA). Our approach consists in determining how much knowledge "extracted" from the CWA can "safely" be used to "complete" a logic program. Our approach leads to novel characterizations, both epistemic and operational, of the well-founded semantics [3] for logic programs and extends that semantics to PDDU. Moreover we show that the continuity of the immediate consequence operator, used for inferring information from the program, is preserved. This is important as this is a major feature of classical PDDU, opposed to classical frameworks like [8]. Negation has already been considered in some deductive databases with uncertainty frameworks. In [13, 14], the stable semantics has been considered, but limited to the case where the underlying uncertainty formalism is probability theory. That semantics has been considered also in [19], where a semi-possibilistic logic has been proposed, 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 PDDU, except than our previous attempt [11]. The semantics defined in [11] is weaker than the one presented in this paper, as in the latter approach more knowledge can be extracted from a program, it has no epistemic characterization, and rely on a less natural management of negation.

In the remaining, we proceed as follows. In the following section, the syntax of PDDU with negation, called normal parametric programs, is given, Section 3 contains the definitions of interpretation and model of a program. In Section 4, we present the fundamental notion of support of a program provided by the CWA with respect to an interpretation. Then we propose novel characterizations of the well-founded semantics and compare our approach with usual semantics. Section 5 concludes.

2 Preliminaries

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 $\pi(A) = p$. The truth-space is given by a complete lattice: atomic formulae are mapped into elements of a *certainty lattice* $\mathcal{L} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$ (a complete lattice), where $\mathcal{T}$ is the set of certainty values, $\preceq$ is a partial order, $\otimes$ and $\oplus$ are the meet and join operators, respectively. With $\bot$ and $\top$ we denote the least and greatest element in $\mathcal{T}$. With $\mathcal{B}(\mathcal{T})$ we denote the set of finite multisets (denoted $\{\!\{ \cdot \}\!\}$) over $\mathcal{T}$. For instance, a typical certainty lattice is $\mathcal{L}_{[0,1]} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$, where $\mathcal{T} = [0, 1]$, $\alpha \preceq \beta$ iff $\alpha \leq \beta$, $\alpha \otimes \beta = \min(\alpha, \beta)$, $\alpha \oplus \beta = \max(\alpha, \beta)$, $\bot = 0$ and $\top = 1$. While the language does not contain function symbols, it contains symbols for families of conjunction ($\mathcal{F}_c$), propagation ($\mathcal{F}_p$) and disjunction functions ($\mathcal{F}_d$), called *combination functions*. Roughly, as we will see below, the conjunction function (e.g. $\otimes$) determines the certainty of the conjunction of $L_1, ..., L_n$ (the body) of a logic program rule like $A \overset{\alpha}{\leftarrow} L_1, ..., L_n$, a propagation function (e.g. $\otimes$) determines how to "propagate" the certainty, resulting from the evaluation of the body $L_1, ..., L_n$, to the head A, by taking into account the certainty α of the implication, while the disjunction function (e.g. $\oplus$) dictates how to combine the certainties in case an atom appears in the heads of several rules (evaluates a disjunction). Examples of conjunction, propagation and disjunction over $\mathcal{L}_{[0,1]}$ are $f_c(x, y) = \min(x, y)$, $f_p(x, y) = xy$, $f_d(x, y) = x + y - xy$. Formally, 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 combination function is monotonic and continuous w.r.t. (with respect to) each one of its arguments. Conjunction and disjunction functions are commutative and associative. Additionally, each kind of function must verify some of the following properties[3]: (i) bounded-

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

above: $f(\alpha_1, \alpha_2) \preceq \alpha_i$, for $i = 1, 2, \forall \alpha_1, \alpha_2 \in \mathcal{T}$; (ii) bounded-below: $f(\alpha_1, \alpha_2) \succeq \alpha_i$, for $i = 1, 2, \forall \alpha_1, \alpha_2 \in \mathcal{T}$; (iii) $f(\{\alpha\}) = \alpha, \forall \alpha \in \mathcal{T}$; (iv) $f(\emptyset) = \bot$; (v) $f(\emptyset) = \top$; and (vi) $f(\alpha, \top) = \alpha, \forall \alpha \in \mathcal{T}$. The following should be satisfied. A conjunction function in $\mathcal{F}_c$ should satisfy properties (i), (iii), (v) and (vi); a propagation function in $\mathcal{F}_p$ should satisfy properties (i) and (vi), while a disjunction function in $\mathcal{F}_d$ should satisfy properties (ii), (iii) and (iv). We also assume that there is a function from $\mathcal{T}$ to $\mathcal{T}$, called *negation function*, denoted $\neg$, that is anti-monotone w.r.t. $\preceq$ and satisfies $\neg\neg\alpha = \alpha, \forall \alpha \in \mathcal{T}$. E.g., in $\mathcal{L}_{[0,1]}$, $\neg\alpha = 1 - \alpha$ is quite typical. Finally, a literal is an atomic formula or its negation.

Definition 1 (Normal parametric program [10]). *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 certainties partially ordered by* $\preceq$, $\otimes$ *is the meet operator and* $\oplus$ *the join operator;*
2. $\mathcal{R}$ *is a finite set of* normal parametric rules *(np-rules), each of which is a statement of the form:* $r : A \overset{\alpha_r}{\leftarrow} L_1, ..., L_n$ *where* A *is an atomic formula,* $L_1, ..., L_n$ *are literals 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 $r : A \overset{\alpha_r}{\leftarrow} L_1, ..., L_n; \langle f_d, f_p, f_c \rangle$ to represent a np-rule in which $f_d \in \mathcal{F}_d$ is the disjunction function associated with $\pi(A)$ and, $f_c \in \mathcal{F}_c$ and $f_p \in \mathcal{F}_p$ are respectively the conjunction and propagation functions associated with r. Note that, by Definition 1, rules with same head must have the same disjunction function associated. The following example illustrates the notion of np-program.

Example 1. Consider an insurance company, which has information about its customers used to determine the risk coefficient of each customer (in order to fix the price of the insurance contracts). The company has: (i) data grouped into a set F of facts; and (ii) a set R of rules. Suppose the company has the following database (which is an np-program $P = F \cup R$), where a value of the risk coefficient may be already known, but has to be re-evaluated (the client may be a new client and his risk coefficient is given by his precedent insurance company). The certainty lattice is $\mathcal{L}_{[0;1]}$, with $f_p(x, y) = xy$.

$$F = \left\{ \begin{array}{lll} \texttt{Experience(John)} & \overset{1}{\leftarrow} 0.7 & \langle \oplus, f_p, \otimes \rangle \\ \texttt{Risk(John)} & \overset{1}{\leftarrow} 0.5 & \langle \oplus, f_p, \otimes \rangle \\ \texttt{Sport\_car(John)} & \overset{1}{\leftarrow} 0.8 & \langle \oplus, f_p, \otimes \rangle \end{array} \right\}$$

$$R = \left\{ \begin{array}{ll} \texttt{Good\_driver(X)} & \overset{1}{\leftarrow} \texttt{Experience(X)}, \neg\texttt{Risk(X)} \;\; \langle \oplus, \otimes, \otimes \rangle \\ \texttt{Risk(X)} & \overset{0.8}{\leftarrow} \texttt{Young(X)}, \;\; \langle \oplus, f_p, \otimes \rangle \\ \texttt{Risk(X)} & \overset{0.8}{\leftarrow} \texttt{Sport\_car(X)} \;\; \langle \oplus, f_p, \otimes \rangle \\ \texttt{Risk(X)} & \overset{1}{\leftarrow} \texttt{Experience(X)}, \neg\texttt{Good\_driver(X)} \;\; \langle \oplus, f_p, \otimes \rangle \end{array} \right\}$$

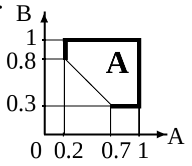
Using another disjunction function associated to the rules with head Risk, such as $f_d(x, y) = x + y - xy$, might have been more appropriate in such an example (i.e. we accumulate the risk factors, rather than take the $\max$ only), but we will use $\oplus$ in order to facilitate the reader's comprehension later on when we compute the semantics of P.

We further define the *Herbrand base* $\mathcal{B}_P$ 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 (P^* is finite). Note that a Datalog program with negation P is equivalent to the np-program constructed by replacing each rule in P of the form $A \leftarrow L_1, ..., L_n$ by the rule $A \overset{t}{\leftarrow} L_1, ..., L_n; \langle \oplus, \otimes, \otimes \rangle$, where the *classical* certainty lattice $\mathcal{L}_{\{t,f\}}$ is considered, where $\mathcal{L}_{\{t,f\}} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$, with $\mathcal{T} = \{t, f\}$, $\preceq$ is defined by $f \preceq t$, $\oplus = \max_{\preceq}$, $\otimes = \min_{\preceq}$, $\neg f = t$ and $\neg t = f$, $\bot = f$ and $\top = t$.

3 Interpretations of programs

The semantics of a program P is determined by selecting a particular interpretation of P in the set of models of P, where an *interpretation* I of an np-program P is a function that assigns to all atoms of the Herbrand base of P a value in $\mathcal{T}$. In Datalog programs, as well as in PDDU, that chosen model is usually the least model of P w.r.t. $\preceq$.[4] Unfortunately, the introduction of negation may have the consequence that some logic programs do not have a unique minimal model, as shown in the following example.

Example 2. Consider the certainty lattice $\mathcal{L}_{[0,1]}$ and the program $P = \{(A \leftarrow \neg B), (B \leftarrow \neg A), (A \leftarrow 0.2), (B \leftarrow 0.3)\}$. Informally, an interpretation I is a model of the program if it satisfies every rule, while I satisfies a rule $X \leftarrow Y$ if $I(X) \succeq I(Y)$[5]. So, this program has an infinite number of models I_x^y, where $0.2 \preceq x \preceq 1$, $0.3 \preceq y \preceq 1$, $y \geq 1 - x$, $I_x^y(A) = x$ and $I_x^y(B) = y$ (those in the A area).



There are also an infinite number of minimal models (those on the thin diagonal line). The minimal models I_x^y are such that $y = 1 - x$. $\square$

Concerning the previous example we may note that the certainty 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 be the answer to a query A to the program proposed in Example 2? There are at least two answers: (i) the certainty 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; (ii) the certainty 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 certainty 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. Applying a usual approach, like the well-founded semantics [18] or the Kripke-Kleene semantics [4], would lead us to choose the conservative solution 1. This was also the approach in our early attempt to deal with normal parametric programs [11]. Such a semantics seems to be too weak, in the sense that it loses some knowledge (e.g. the value of A should be at least 0.2). In this paper we address solution 2.

To this end, we propose to rely on $\mathcal{T} \times \mathcal{T}$. Any element of $\mathcal{T} \times \mathcal{T}$ is denoted by $[a; b]$ and interpreted as an interval on $\mathcal{T}$, i.e. $[a; b]$ *is interpreted as the set of elements* $x \in \mathcal{T}$ *such that* $a \preceq x \preceq b$. For instance, turning back to Example 2 above, in the intended model of P, the certainty of A is "approximated" with $[0.2; 0.7]$, i.e. the certainty of A lies in between 0.2 and 0.7 (similarly for B). Formally, given a complete lattice $\mathcal{L} = \langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$, we

[4] $\preceq$ is extended to the set of interpretations as follows: $I \preceq J$ iff for all atoms A, $I(A) \preceq J(A)$.

[5] Roughly, $X \leftarrow Y$ dictates that "X should be at least as true as Y.

construct a *bilattice* over $\mathcal{T} \times \mathcal{T}$, according to a well-known construction method (see [3, 6]). We recall that a bilattice is a triple $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$, where $\mathcal{B}$ is a nonempty set and $\preceq_t$, $\preceq_k$ are both partial orderings giving to $\mathcal{B}$ the structure of a lattice with a top and a bottom [6]. We consider $\mathcal{B} = \mathcal{T} \times \mathcal{T}$ with orderings:

– the *truth ordering* $\preceq_t$, where $[a_1; b_1] \preceq_t [a_2; b_2]$ iff $a_1 \preceq a_2$ and $b_1 \preceq b_2$;
– the *knowledge ordering* $\preceq_k$, where $[a_1; b_1] \preceq_k [a_2; b_2]$ iff $a_1 \preceq a_2$ and $b_2 \preceq b_1$.

The intuition of those orders is that truth increases if the interval contains greater values (e.g. $[0.1; 0.4] \preceq_t [0.2; 0.5]$), whereas the knowledge increases when the interval (i.e. in our case the approximation of a certainty value) becomes more precise (e.g. $[0.1; 0.4] \preceq_k [0.2; 0.3]$, i.e. we have more knowledge). The least and greatest elements of $\mathcal{T} \times \mathcal{T}$ are respectively (i) $\mathtt{f} = [\bot; \bot]$ (false) and $\mathtt{t} = [\top; \top]$ (true), w.r.t. $\preceq_t$; and (ii) $\bot = [\bot; \top]$ (unknown – the less precise interval, i.e. the atom's certainty value is unknown) and $\top = [\top; \bot]$ (inconsistent – the empty interval) w.r.t. $\preceq_k$. The meet, join and negation on $\mathcal{T} \times \mathcal{T}$ w.r.t. both orderings are defined by extending the meet, join and negation from $\mathcal{T}$ to $\mathcal{T} \times \mathcal{T}$ in the natural way: let $[a_1; b_1], [a_2; b_2] \in \mathcal{T} \times \mathcal{T}$, then

– $[a_1; b_1] \otimes^t [a_2; b_2] = [a_1 \otimes a_2; b_1 \otimes b_2]$ and $[a_1; b_1] \oplus^t [a_2; b_2] = [a_1 \oplus a_2; b_1 \oplus b_2]$;
– $[a_1; b_1] \otimes^k [a_2; b_2] = [a_1 \otimes a_2; b_1 \oplus b_2]$ and $[a_1; b_1] \oplus^k [a_2; b_2] = [a_1 \oplus a_2; b_1 \otimes b_2]$;
– $\neg[a_1; b_1] = [\neg b_1; \neg a_1]$.

$\otimes^t$ and $\oplus^t$ ($\otimes^k$ and $\oplus^k$) denote the meet and join operations on $\mathcal{T} \times \mathcal{T}$ w.r.t. the truth (knowledge) ordering, respectively. For instance, taking $\mathcal{L}_{[0,1]}$, $[0.1; 0.4] \oplus^t [0.2; 0.5] = [0.2; 0.5]$, $[0.1; 0.4] \otimes^t [0.2; 0.5] = [0.1; 0.4]$, $[0.1; 0.4] \oplus^k [0.2; 0.5] = [0.2; 0.4]$, $[0.1; 0.4] \otimes^k [0.2; 0.5] = [0.1; 0.5]$ and $\neg[0.1; 0.4] = [0.6; 0.9]$. Finally, we extend in a similar way the combination functions from $\mathcal{T}$ to $\mathcal{T} \times \mathcal{T}$. Let f_c (resp. f_p and f_d) be a conjunction (resp. propagation and disjunction) function over $\mathcal{T}$ and $[a_1; b_1], [a_2; b_2] \in \mathcal{T} \times \mathcal{T}$:

– $f_c([a_1; b_1], [a_2; b_2]) = [f_c(a_1, a_2); f_c(b_1, b_2)]$;
– $f_p([a_1; b_1], [a_2; b_2]) = [f_p(a_1, a_2); f_p(b_1, b_2)]$;
– $f_d([a_1; b_1], [a_2; b_2]) = [f_d(a_1, a_2); f_d(b_1, b_2)]$.

It is easy to verify that these extended combination functions preserve the original properties of combination functions.The following theorem holds.

Theorem 1. *Consider $\mathcal{T} \times \mathcal{T}$ with the orderings $\preceq_t$ and $\preceq_k$. Then*

1. *$\otimes^t, \oplus^t, \otimes^k, \oplus^k$ and the extensions of combination functions are continuous (and, thus, monotonic) w.r.t. $\preceq_t$ and $\preceq_k$;*
2. *any extended negation function is monotonic w.r.t. $\preceq_k$;*
3. *if the negation function satisfies the de Morgan laws, i.e. $\forall a, b \in \mathcal{T}. \neg(a \oplus b) = \neg a \otimes \neg b$ then the extended negation function is continuous w.r.t. $\preceq_k$.*

Proof. We proof only the last item, as the others are immediate. Consider a chain of intervals $x_0 \preceq_k x_1 \preceq_k \ldots$, where $x_j = [a_j; b_j]$ with $a_j, b_j \in \mathcal{T}$. To show the continuity of the extended negation function w.r.t. $\preceq_k$, we show that $\neg \oplus^k_{j \geq 0} x_j = \oplus^k_{j \geq 0} \neg x_j$:
$\neg \oplus^k_{j \geq 0} x_j = \neg[\oplus_{j \geq 0} a_j; \otimes_{j \geq 0} b_j] = [\neg \otimes_{j \geq 0} b_j; \neg \oplus_{j \geq 0} a_j] = [\oplus_{j \geq 0} \neg b_j; \otimes_{j \geq 0} \neg a_j]$
$= \oplus^k_{j \geq 0} [\neg b_j; \neg a_j] = \oplus^k_{j \geq 0} \neg [a_j; b_j] = \oplus^k_{j \geq 0} \neg x_j$.
We can now extend interpretations over $\mathcal{T}$ to the above specified "interval" bilattice.

Definition 2 (Approximate interpretation). *Let P be an np-program. An* approximate interpretation *of P is a total function I from the Herbrand base $\mathcal{B}_P$ to the set $\mathcal{T} \times \mathcal{T}$. The set of all the approximate interpretations of P is denoted $\mathcal{C}_P$.*

Intuitively, as anticipated, assigning the logical value $[a; b]$ to an atom A means that the exact certainty value of A lies in between a and b with respect to $\preceq$. Note that $I(A) = [a; b]$ can also be understood as the specification of a *lower bound and upper bound constraint* on the admissible certainty values of A. Our goal will be to determine for each atom of the Herbrand base of P the most precise interval that can be inferred, i.e. the maximal value with respect to $\preceq_k$, i.e. maximal knowledge.

At first, we extend the two orderings on $\mathcal{T} \times \mathcal{T}$ to the set of approximate interpretations $\mathcal{C}_P$ in a usual way: let I_1 and I_2 be in $\mathcal{C}_P$, then (i) $I_1 \preceq_t I_2$ iff $I_1(A) \preceq_t I_2(A)$, for all ground atoms A; and (ii) $I_1 \preceq_k I_2$ iff $I_1(A) \preceq_k I_2(A)$, for all ground atoms A. Under these two orderings $\mathcal{C}_P$ becomes a complete bilattice. The meet and join operations over $\mathcal{T} \times \mathcal{T}$ for both orderings are extended to $\mathcal{C}_P$ in the usual way (e.g. for any atom A, $(I \oplus^k J)(A) = I(A) \oplus^k J(A)$). Negation is extended similarly, for any atom A, $\neg I(A) = I(\neg A)$, and approximate interpretations are extended to $\mathcal{T}$, for any $\alpha \in \mathcal{T}$, $I(\alpha) = [\alpha; \alpha]$. At second, we identify the models of a program. The definition extends the one given in [10] to intervals.

Definition 3 (Models of a logic program). *Let P be an np-program and let I be an approximate interpretation of P.*

1. *I satisfies a ground np-rule $r : A \overset{\alpha_r}{\leftarrow} L_1, ..., L_n; \langle f_d, f_p, f_c \rangle$ in P, denoted $\models_I r$, iff $f_p([\alpha_r; \alpha_r], f_c(\{I(L_1), \ldots, I(L_n)\})) \preceq_t I(A)$;*
2. *I is a model of P, or I satisfies P, denoted $\models_I P$, iff for all atoms $A \in \mathcal{B}_P$, $f_d(X) \preceq_t I(A)$ where f_d is the disjunction function associated with $\pi(A)$ and*

$$X = \{f_p([\alpha_r; \alpha_r], f_c(\{I(L_1), \ldots, I(L_n)\})) : A \overset{\alpha_r}{\leftarrow} L_1, ..., L_n; \langle f_d, f_p, f_c \rangle \in P^*\}.$$

At third, among all possible models of an np-program, we have now to specify which one is the intended model. The characterization of that model will require the definition of an immediate consequence operator that will be used to infer knowledge from a program. That operator is a simple extension from $\mathcal{T}$ to $\mathcal{T} \times \mathcal{T}$ of the immediate consequence operator defined in [10] to give semantics to classical PDDU.

Definition 4. *Let P be any np-program. The* immediate consequence operator *T_P is a mapping from $\mathcal{C}_P$ to $\mathcal{C}_P$, defined as follows: for every interpretation I in $\mathcal{C}_P$, for every ground atom A, $T_P(I)(A) = f_d(X)$, where f_d is the disjunction function associated with $\pi(A)$ and $X = \{f_p([\alpha_r; \alpha_r], f_c(\{I(L_1), \ldots, I(L_n)\})) : A \overset{\alpha_r}{\leftarrow} L_1, ..., L_n; \langle f_d, f_p, f_c \rangle \in P^*\}$.*

Note that from the property iv of combination functions satisfied by all disjunction functions, it follows that if an atom A does not appear as the head of a rule, then $T_P(I)(A) = \mathtt{f}$. Note also that any fixpoint of T_P is a model of P. We have

Theorem 2. *For any np-program P, T_P is monotonic and, if the de Morgan laws hold, continuous w.r.t. $\preceq_k$.*

Proof. The proof of monotonicity is easy. To prove the continuity w.r.t. $\preceq_k$, consider a chain of interpretations $I_0 \preceq_k I_1 \preceq_k \ldots$. We show that for any $A \in \mathcal{B}_P$,

$$T_P(\oplus^k_{j \geq 0} I_j)(A) = \oplus^k_{j \geq 0} T_P(I_j)(A) \tag{1}$$

As $\mathcal{C}_P$ is a complete lattice, the sequence $I_0 \preceq_k I_1 \preceq_k \ldots$ has a least upper bound, say $\bar{I} = \oplus^k_{j \geq 0} I_j$. For any $B \in \mathcal{B}_P$, we have $\oplus^k_{j \geq 0} I_j(B) = \bar{I}(B)$ and, from Theorem 1,

$\oplus_{j\geq 0}^{k} I_j(\neg B) = \oplus_{j\geq 0}^{k} \neg I_j(B) = \neg \oplus_{j\geq 0}^{k} I_j(B) = \neg \bar{I}(B)$ and, thus, for any literal or certainty value L,

$$\oplus_{j\geq 0}^{k} I_j(L) = \bar{I}(L) \tag{2}$$

Now, consider the finite set (P^* is finite) of all ground rules $r_1, \ldots, r_k$ having A as head, where $r_i = A \overset{\alpha_i}{\leftarrow} L_1^i, ..., L_{n_i}^i; \langle f_d, f_p^i, f_c^i \rangle)$. Let us evaluate the left hand side of Equation 1. $T_P(\oplus_{j\geq 0}^{k} I_j)(A) = T_P(\bar{I})(A) = f_d(\{f_p^i([\alpha_i;\alpha_i], f_c^i(\{\bar{I}(L_1^i), \ldots, \bar{I}(L_{n_i}^i\})) \colon 0 \leq i \leq k\})$. On the other hand side, $\oplus_{j\geq 0}^{k} T_P(I_j)(A) = \oplus_{j\geq 0}^{k} f_d(\{f_p^i([\alpha_i;\alpha_i], f_c^i(\{I_j(L_1^i), \ldots, I_j(L_{n_i}^i\})) \colon 0 \leq i \leq k\})$. But, f_d, f_p^i and f_c^i are continuous and, thus, by Equation 2

$$\begin{aligned}
\oplus_{j\geq 0}^{k} T_P(I_j)(A) &= f_d(\{\oplus_{j\geq 0}^{k} \{f_p^i([\alpha_i;\alpha_i], f_c^i(\{I_j(L_1^i), \ldots, I_j(L_{n_i}^i\})) \colon 0 \leq i \leq k\}\}) \\
&= f_d(\{f_p^i([\alpha_i;\alpha_i], \oplus_{j\geq 0}^{k}\{f_c^i(\{I_j(L_1^i), \ldots, I_j(L_{n_i}^i\})\}) \colon 0 \leq i \leq k\}) \\
&= f_d(\{f_p^i([\alpha_i;\alpha_i], f_c^i(\{\oplus_{j\geq 0}^{k} I_j(L_1^i), \ldots, \oplus_{j\geq 0}^{k} I_j(L_{n_i}^i\})) \colon 0 \leq i \leq k\}) \\
&= f_d(\{f_p^i([\alpha_i;\alpha_i], f_c^i(\{\bar{I}(L_1^i), \ldots, \bar{I}(L_{n_i}^i\})) \colon 0 \leq i \leq k\})
\end{aligned}$$

Therefore, Equation 1 holds and, thus, T_P is continuous.

4 Semantics of normal logic programs

Usually, the semantics of a normal logic program is the least model of the program with respect to the knowledge ordering. That model always exists and coincides with the the least fixed-point of T_P with respect to $\preceq_k$ (which exists as T_P is monotonic w.r.t. $\preceq_k$). Note that this least model with respect to $\preceq_k$ corresponds to an extension of the classical Kripke-Kleene semantics [4] of Datalog programs with negation to normal parametric programs: if we restrict our attention to Datalog with negation, then we have to deal with four values $[f;f], [t;t], [f;t]$ and $[t;f]$ that correspond to the truth values *false*, *true*, *unknown* and *inconsistent*, respectively. Then, our bilattice coincides with Belnap's logic [1] and for any Datalog program with negation P, the least fixed-point of T_P w.r.t. $\preceq_k$ is a model of P that coincides with the Kripke-Kleene semantics of P.

To illustrate the different notions introduced in the paper, we rely on the Example 3.

Example 3 (Running example). The certainty lattice is $\mathcal{L}_{[0,1]}$ and the np-program is $P = \{(A \overset{1}{\leftarrow} B, 0.6; \langle \oplus, \otimes, \otimes \rangle), (B \overset{1}{\leftarrow} B; \langle \oplus, \otimes, \otimes \rangle), (A \overset{1}{\leftarrow} 0.3; \langle \oplus, \otimes, \otimes \rangle)\}$. □

The table below shows how the Kripke-Kleene semantics, KK_P, of the running Example 3 is computed (as the iterated fixed-point of T_P, starting from $I_0 = I_\perp$, the $\preceq_k$ minimal interpretation that maps any $A \in \mathcal{B}_P$ to $[\perp;\top]$, and $I_{n+1} = T_P(I_n)$):

	I_0	I_1	$I_2 = I_1 = KK_P$
A	[0; 1]	[0.3; 0.6]	[0.3; 0.6]
B	[0; 1]	[0; 1]	[0; 1]

In that model, which is minimal w.r.t. $\preceq_k$ and contains only the knowledge provided by P, the certainty of B lies between 0 and 1, i.e. is unknown, and the certainty of A then lies between 0.3 and 0.6. As well known, that semantics is usually considered as too weak. We propose to consider the *Closed World Assumption* (CWA) to complete our knowledge (the CWA assumes that all atoms whose value cannot be inferred from the program are false by default). This is done by defining the notion of *support*, introduced in [12], of a program with respect to an interpretation. Given a program P and an interpretation I, the support of P w.r.t. I, denoted $C_P(I)$, determines in a principled way how much *false* knowledge, i.e. how much knowledge provided by the CWA, can "safely" be joined to I with respect to the program P. Roughly speaking, a part of the CWA is an interpretation J such that $J \preceq_k \mathtt{I_f}$, where $\mathtt{I_f}$ maps any $A \in \mathcal{B}_P$ to $[\bot; \bot]$, and we consider that such an interpretation can be safely added to I if $J \preceq_k T_P(I \oplus^k J)$, i.e. if J does not contradict the knowledge represented by P and I.

Definition 5. *The* support of an np-program P w.r.t. an interpretation I, *denoted* $C_P(I)$, *is the maximal interpretation J w.r.t. $\preceq_k$ such that $J \preceq_k \mathtt{I_f}$ and $J \preceq_k T_P(I \oplus^k J)$.*

It is easy to note that $C_P(I) = \bigoplus^k \{J \mid J \preceq_k \mathtt{I_f} \text{ and } J \preceq_k T_P(I \oplus^k J)\}$. The following theorem provides an algorithm for computing the support.

Theorem 3. *$C_P(I)$ coincides with the iterated fixpoint of the function $F_{P,I}$ beginning the computation with $I_\mathtt{f}$, where $F_{P,I}(J) = I_\mathtt{f} \otimes^k T_P(I \oplus^k J)$.*

From Theorems 1 and 2, it can be shown that $F_{P,I}$ is monotone and, if the de Morgan laws hold, continuous w.r.t. $\preceq_k$. It follows that the iteration of the function $F_{P,I}$ starting from $I_\mathtt{f}$ decreases w.r.t. $\preceq_k$. We will refer to C_P as the *closed world operator*.

Corollary 1. *Let P be an np-program. The closed world operator C_P is monotone and, if the de Morgan laws hold, continuous w.r.t. the knowledge order $\preceq_k$.*

The table below shows the computation of $C_P(KK_P)$, i.e. the additional knowledge that can be considered using the CWA on the Kripke-Kleene semantics KK_P of the running Example 3 ($I = KK_P$, $J_0 = I_\mathtt{f}$ and $J_{n+1} = F_{P,I}(J_n)$):

	J_0	J_1	$J_2 = J_1 = C_P(KK_P)$
A	$[0; 0]$	$[0; 0.3]$	$[0; 0.3]$
B	$[0; 0]$	$[0; 0]$	$[0; 0]$

The interpretation $C_P(KK_P)$ asserts that, according to the CWA and with respect to P and KK_P, the certainty of A should be at most 0.3 and the certainty of B exactly 0. We have now two ways to infer information from an np-program P and an approximate interpretation I: using T_P and using C_P. To maximize the knowledge derived from P and the CWA, but without introducing any other extra knowledge, we propose to choose the least model of P containing its own support, i.e. that cannot be completed anymore according to the CWA, as the semantics of P. This consideration leads to the following epistemic definition of semantics of a program P.

Definition 6. *The approximate well-founded semantics of an np-program P, denoted W_P, is the least model I of P w.r.t. $\preceq_k$ such that $C_P(I) \preceq_k I$.*

Now we provide a fixpoint characterization and, thus, a way of computation of the approximate well-founded semantics. It is based on an operator, called approximate well-founded operator, that combines the two operators that have been defined above. Given an interpretation I, we complete it with its support provided by the CWA, and then activate the rules of the program on the obtained interpretation using the immediate consequence operator.

Definition 7. *Let P be an np-program. The approximate well-founded operator, denoted AW_P, takes in input an approximate interpretation $I \in \mathcal{C}_P$ and returns $AW_P(I) \in \mathcal{C}_P$ defined by $AW_P(I) = T_P(I \oplus^k C_P(I))$.*

From [12], the following theorems can be shown.

Theorem 4. *Let P be an np-program. Any fixed-point I of AW_P is a model of P.*

Using the properties of monotonicity and continuity of T_P and C_P w.r.t. the knowledge order $\preceq_k$ over $\mathcal{C}_P$, from the fact that $\mathcal{C}_P$ is a complete lattice w.r.t. $\preceq_k$, by the well-known Knaster-Tarski theorem, it follows that

Theorem 5. *Let P be an np-program. The approximate well-founded operator AW_P is monotone and, if the de Morgan laws hold, continuous w.r.t. the knowledge order $\preceq_k$. Therefore, AW_P has a least fixed-point w.r.t. the knowledge order $\preceq_k$. Moreover that least fixpoint coincides with the approximate well-founded semantics W_P of P.*

The table below shows the computation of W_P of Example 3 ($I_0 = I_\perp$ and $I_{n+1} = AW_P(I_n)$). The certainty of A is 0.3 and the certainty of B is 0. Note that $KK_P \preceq_k W_P$, i.e. the well-founded semantics contains more knowledge than the Kripke-Kleene semantics that was completed with some default knowledge from the CWA.

	I_0	$C_P(I_0)$	I_1	$C_P(I_1)$	$I_2 = I_1 = W_P$
A	$[0;1]$	$[0;0.3]$	$[0.3;0.3]$	$[0;0.3]$	$[0.3;0.3]$
B	$[0;1]$	$[0;0]$	$[0;0]$	$[0;0]$	$[0;0]$

We conclude with re-considering the example 1 seen at the beginning.

Example 4. Consider the program $P = R \cup F$ given in Example 1. The computation of the approximate well-founded semantics W_P of P is as follows[6]:

	R(J)	S(J)	Y(J)	G(J)	E(J)
$I_0 = I_\perp$	$[0;1]$	$[0;1]$	$[0;1]$	$[0;1]$	$[0;1]$
$I_1 = AW_P(I_0)$	$[0.5;0.7]$	$[0.8;0.8]$	$[0.0;0.0]$	$[0.0;0.7]$	$[0.7;0.7]$
$I_2 = AW_P(I_1)$	$[0.64;0.7]$	$[0.8;0.8]$	$[0.0;0.0]$	$[0.3;0.5]$	$[0.7;0.7]$
$I_3 = AW_P(I_2)$	$[0.64;0.7]$	$[0.8;0.8]$	$[0.0;0.0]$	$[0.3;0.36]$	$[0.7;0.7]$
$W_P = I_3 = AW_P(I_3)$	$[0.64;0.7]$	$[0.8;0.8]$	$[0.0;0.0]$	$[0.3;0.36]$	$[0.7;0.7]$

with

	R(J)	S(J)	Y(J)	G(J)	E(J)
$C_P(I_0)$	$[0.0;0.7]$	$[0.0;0.8]$	$[0.0;0.0]$	$[0.0;0.7]$	$[0.0;0.7]$
$C_P(I_1)$	$[0.0;0.7]$	$[0.0;0.8]$	$[0.0;0.0]$	$[0.0;0.7]$	$[0.0;0.5]$
$C_P(I_2)$	$[0.0;0.7]$	$[0.0;0.8]$	$[0.0;0.0]$	$[0.0;0.7]$	$[0.0;0.36]$
$C_P(I_3)$	$[0.0;0.7]$	$[0.0;0.8]$	$[0.0;0.0]$	$[0.0;0.7]$	$[0.0;0.36]$

which establishes that `John`'s degree of `Risk` is in between $[0.64, 0.7]$. $\square$

Finally, our approach captures and extends the usual semantics of logic programs.

Theorem 6. *If we restrict our attention to PDDU, then for any program P the approximate well-founded semantics W_P assigns exact values to all atoms and coincides with the semantics of P proposed in [10].*

Theorem 7. *If we restrict our attention to Datalog with negation, then we have to deal with Belnap's bilattices [1] and for any Datalog program with negation P,*

- *any stable model [5] of P is a fixpoint of AW_P, and*
- *the approximate well-founded semantics W_P coincides with the well-founded semantics of P [18].*

Note that any stable model is a model containing its own support, i.e. a model that cannot be completed anymore according to the CWA. Note also that our approach gives a simple way to verify the well-known result that, for any Datalog programs with negation P, the Kripke-Kleene semantics of P gives less knowledge than the well-founded semantics of P, i.e. is smaller w.r.t. the knowledge order.

[6] For ease of presentation, we use the first letter of predicates and constants only.

5 Conclusions

We define novel characterizations, both epistemic and operational, of the well-founded semantics in the general framework of PDDU [10], an unifying umbrella for many existing approaches towards the manipulation of uncertainty in logic programs, that we extend with non-monotonic (default) negation. Main features of our extension are (i) dealing with uncertain and incomplete knowledge, atoms are assigned approximation of uncertainty values; (ii) the CWA is used to complete the knowledge in order to infer the most precise approximations as possible relying on a natural management of negation; (iii) that the continuity of the immediate consequence operator is preserved (which is a major feature of the classical PDDU framework); and (iv) our approach captures and extends to PDDU with negation not only the semantics proposed in [10] for PDDU, but also the usual semantics of Datalog with negation: the well-founded semantics and the Kripke-Kleene semantics.

References

1. N. D. Belnap. How a computer should think. In Gilbert Ryle, editor, *Contemporary aspects of philosophy*, pages 30–56. Oriel Press, Stocksfield, GB, 1977.
2. D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In *Proc. of the 8th Int. Conf. on Logic Programming (ICLP-91)*, pages 581–595, 1991.
3. M. Fitting. The family of stable models. *J. of Logic Programming*, 17:197–225, 1993.
4. M. Fitting. A Kripke-Kleene-semantics for general logic programs. *J. of Logic Programming*, 2:295–312, 1985.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th Int. Conf. on Logic Programming*, pages 1070–1080, 1988.
6. M. L. Ginsberg. Multi-valued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.
7. M. Kifer and A. Li. On the semantics of rule-based expert systems with uncertainty. In *Proc. of the Int. Conf. on Database Theory (ICDT-88)*, in LNCS 326, pages 102–117, 1988.
8. M. Kifer and V.S. Subrahmanian. Theory of generalized annotaded logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
9. L. V.S. Lakshmanan and N. Shiri. Probabilistic deductive databases. In *Int. Logic Programming Symposium*, pages 254–268, 1994.
10. L. V.S. Lakshmanan and N. Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
11. Y. Loyer and U. Straccia. The well-founded semantics in normal logic programs with uncertainty. In *Proc. of the 6th Int. Symposium on Functional and Logic Programming (FLOPS-2002)*, in LNCS 2441, pages 152–166, 2002.
12. Y. Loyer and U. Straccia. The well-founded semantics of logic programs over bilattices: an alternative characterisation. Technical Report ISTI-2003-TR-05, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, 2003. Submitted.
13. T. Lukasiewicz. Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics, in LNCS, 2173, pages 336–350, 2001.
14. R. Ng and V.S. Subrahmanian. Stable model semantics for probabilistic deductive databases. In *Proc. of the 6th Int. Symposium on Methodologies for Intelligent Systems (ISMIS-91)*, in LNAI 542, pages 163–171, 1991.
15. R. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1993.
16. E. 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. *J. of Logic Programming*, 4(1):37–53, 1986.
18. A. van Gelder, K. A. Ross, and J. S. Schlimpf. The well-founded semantics for general logic programs. *J. of the ACM*, 38(3):620–650, January 1991.
19. G. Wagner. Negation in fuzzy and possibilistic logic programs. In T. Martin and F. Arcelli, editors, *Logic programming and Soft Computing*. Research Studies Press, 1998.