Possible Worlds Semantics for Probabilistic Logic Programs

Alex Dekhtyar, Michael I. Dekhtyar

Abstract. In this paper we consider a logic programming framework for reasoning about imprecise probabilities. In particular, we propose a new semantics, for the Probabilistic Logic Programs (p-programs) of Ng and Subrahmanian. P-programs represent imprecision using probability intervals. Our semantics, based on the possible worlds semantics, considers all point probability distributions that satisfy a given p-program. In the paper, we provide the exact characterization of such models of a p-program. We show that the set of models of a p-program cannot, in general case, be described by single intervals associated with atoms of the program. We provide algorithms for efficient construction of this set of models and study their complexity.

1 Introduction

Probabilities quantize our knowledge about possibilities. Imprecise probabilities represent our uncertainty about such quantization. They arise from incomplete data or from human unsureness. They occur in the analyses of survey responses, in the use of GIS data, and risk assessment, to cite a few application domains. The importance of imprecise probabilities has been observed by numerous researchers in the past 10-15 years [14, 3] and lead to the establishment of the Imprecise Probabilities Project [6]. The appeal of standard, or point, probability theory is its clarity. Given the need for imprecision, there are many models: second-order distributions, belief states or lower envelopes, intervals, and others. Among them, probability intervals as the means of representing imprecision are the simplest extension of the traditional probability models. Even then, a variety of different explanations of what it means for a probability of an event e to be expressed as an interval $[a, b] \subseteq [0, 1]$ have been proposed in the past decade and a half [14, 2, 15, 3]. Among them, the possible worlds approach introduced for probability distributions by De Campos, Huete and Moral [3] and extended to Kolmogorov probability theory by Weichselberger [15] is, probably, the most appealing from the point of view of the origins of imprecision. According to [3, 15], there is a single true *point* probability distribution underlying a collection of random variables or events. The imprecision, expressed in terms of probability intervals stems from our inability to establish what this true distribution *is*. Thus, interval probabilities are constraints on the set of possible point probability distributions that can be true.

Previous logic programming frameworks addressing the issue of imprecision in known probabilities [8, 7, 9, 4, 5] had taken similar approaches to interpreting probability intervals, but have stopped short of considering *precise descriptions* of sets of point probabilities as the semantics of probabilistic logic programs. In this paper, we seek to extend one such logic programming framework, Probabilistic Logic Programs (p-programs), introduced by Ng and Subrahmanian[9], to capture the exact set of point probabilities that satisfy a p-program.

The main contributions of this paper are as follows. First, we describe the possible worlds semantics for a simple logic programming language in which probability intervals are associated with each atom in a program (simple p-programs) (Section 2). The syntax of the language and its model theory are from [9], however, we show that the fixpoint semantics described there does not capture precisely the set of all point probability models of a program (Section 2.2). We then proceed to describe this set of models formally (Section 3.1), and provide an explicit construction for it (Section 3.2), complete with algorithms for implementing this constructions. We show that associating single intervals with atoms of p-programs is not sufficient to capture their model-theoretic semantics: one has to consider unions of open, closed and semi-closed intervals. We also show that while the size of such description of the set of models of a simple p-program can be, in the worst case exponential in the size of the program, our algorithm GenModT for its construction, works in an efficient manner.

2 Probabilistic Logic Programs

In this section we describe a simplified version of the Probabilistic Logic Programs of Ng and Subrahmanian [9]. Let L be some first order language containing infinitely many variable symbols, finitely many predicate symbols and no function symbols. Let $B_L = \{A_1, \dots, A_N\}$ be the Herbrand base of L. A *p-annotated* atom, is an expression $A : \mu$ where $A \in B_L$, and $\mu = [\alpha, \beta] \subseteq [0, 1]$.

P-annotated atoms represent probabilistic information. Every atom in B_L is assumed to represent an (uncertain) event or statement. A *p-annotated atom* $A : [\alpha, \beta]$ is read as "the probability of the event corresponding to A to occur (have occurred) lies in the interval $[\alpha, \beta]$". Probabilistic Logic Programs (p-programs) are constructed from *p-annotated* formulas as follows. Let $A, A_1, \dots A_n$ be some atoms and $\mu, \mu_1, \dots, \mu_n$ be subintervals of $[0, 1]$ (also called *annotations*). Then, a simple *p-clause* is an expression of the form $A : \mu \longleftarrow A_1 : \mu_1 \wedge \dots \wedge A_n : \mu_n$ (if $n = 0$, as usual, the *p-clause* $A : \mu \longleftarrow$ is referred to as a *fact*). A simple Probabilistic Logic Program (*p-program*) is a finite collection of simple *p-clauses*.

2.1 Model Theory and Fixpoint Semantics

The model theory assumes that in real world each atom from B_L is either true or false. However, the observer does not have exact information about the real world, and expresses his/her uncertainty in a form of a probability range. Given B_L, a world probability density function KI is defined as $KI : 2^{B_L} \to [0, 1]$,

$\sum_{W \subseteq B_L} KI(W) = 1$. Each subset W of B_L is considered to be a *possible world* and KI associates a point probability with it. A *probabilistic interpretation* (*p-interpretation*) I is defined on B_L as follows: $I : B_L \to [0,1]$, $I(A) = \sum_{A \in W} KI(W)$. P-interpretations assign probabilities to individual atoms of B_L by adding up the probabilities of all worlds in which a given atom is true. P-interpretations specify the model-theoretic semantics of p-programs. Given a p-interpretation I, the following definitions of satisfaction are given:
– $I \models A : \mu$ **iff** $I(A) \in \mu$;
– $I \models A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ **iff** $(\forall 1 \leq i \leq n)(I \models A_i : \mu_i)$;
– $I \models A : \mu \longleftarrow A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ **iff** either $I \models A : \mu$ or $I \not\models A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$.

Now, given a p-program P, $I \models P$ iff for all p-clauses $C \in P$ $I \models C$. Let $Mod(P)$ denote the set of all p-interpretations that satisfy p-program P. It is convenient to view a single p-interpretation I as a point $(I(A_1), \ldots, I(A_N))$ in N-dimensional unit cube E^N. Then, $Mod(P)$ can be viewed as a subset of E^N. P is called *consistent* iff $Mod(P) \neq \emptyset P$, otherwise P is called *inconsistent*.

Fixpoint semantics of simple p-programs is defined in terms of functions that assign intervals of probability values to atoms of B_L. An *atomic function* is a mapping $f : B_L \longrightarrow \mathcal{C}[0,1]$ where $\mathcal{C}[0,1]$ denotes the set of all closed subintervals of $[0,1]$. Generally, an atomic function f describes a closed parallelepiped in N-dimensional space: a family of p-interpretations $\mathcal{I}(f) = \{I | (\forall A \in B_L)(I(A) \in f(A))\}$ is associated.

The set of all atomic functions over B_L forms a complete lattice $\mathcal{FF}$ w.r.t. the subset inclusion: $f_1 \leq f_2$ iff $(\forall A \in B_L)(f_1(A) \supseteq f_2(A))$. The bottom element $\perp$ of this lattice is the atomic function that assigns $[0,1]$ interval to all atoms, and the top element $\top$ is the atomic function that assigns $\emptyset$ to all atoms.

Given a simple p-program P the fixpoint operator $T_P : \mathcal{FF} \longrightarrow \mathcal{FF}$ is defined as $T_P(f)(A) = \cap M_A$, where $M_A = \{\mu | A : \mu \longleftarrow B_1 : \mu_1 \wedge \ldots \wedge B_n : \mu_n \in P \text{ and } (\forall 1 \leq i \leq n)(f(B_i) \subseteq \mu_i)\}$. Ng and Subrahmanian show that this operator is monotonic [9]. The iterations of T_P are defined in a standard way: (i) $T_P^0 = \perp$; (ii) $T_P^{\alpha+1} = T_P(T_P^{\alpha})$, where $\alpha + 1$ is the successor ordinal whose immediate predecessor is α; (iii) $T_P^{\lambda} = \sqcup\{T_P^{\alpha} | \alpha \leq \lambda\}$, where λ is a limit ordinal. Ng and Subrahmanian show that, the least fixpoint $lfp(T_P)$ of the T_P operator is reachable after a finite number of iterations ([9], Lemma 4). They also show that if a p-program P is consistent, then $\mathcal{I}(lfp(T_P))$ contains $Mod(P)$ ([9] Theorem 5, Claim (i)).

$a : [0.2, 0.4] \longleftarrow .$	(1)
$b : [0.2, 0.5] \longleftarrow .$	(2)
$b : [0.2, 0.3] \longleftarrow a : [0.2, 0.3].$	(3)
$b : [0.4, 0.5] \longleftarrow a : [0.3, 0.4].$	(4)

Program P_1

$a : [0.2, 0.4] \longleftarrow .$	(1)
$b : [0.3, 0.5] \longleftarrow .$	(2)
$b : [0.6, 0.7] \longleftarrow a : [0.2, 0.3].$	(3)
$b : [0.6, 0.7] \longleftarrow a : [0.3, 0.4].$	(4)

Program P_2

Fig. 1. Sample P-programs.

2.2 Fixpoint is not enough

At the same time, the inverse of the last statement, is not true, as evidenced by the following examples. First, consider p-program P_1 shown in Figure 1.

Proposition 1. *There exists a p-interpretation* I *s. t.* $I \in \mathcal{I}(lfp(T_{P_1}))$ *but* $I \not\models P_1$.

Proof. It is easy to see that neither rule (3) nor rule (4) will fire during the computation of the least fixpoint. Indeed, $T^1_{P_1}(a) = [0.2, 0.4]$ and $T^1_{P_1}(b) = [0.2, 0.5]$ based on clauses (1) and (2). However, at the next step, as $[0.2, 0.4] \not\subseteq [0.2, 0.3]$, rule (3) will not fire and as $[0.2, 0.4] \not\subseteq [0.3, 0.4]$, rule (4) will not fire. Therefore, $lfp(T_{P_1}) = T^1_{P_1}$.

Now, consider p-interpretation I, such that $I(a) = 0.2$ and $I(b) = 0.35$. Clearly, $I \in \mathcal{I}(lfp(T_{P_1}))$. However, $I \not\models P_1$. Indeed, as $I(a) = 0.2 \in [0.2, 0.3]$, I satisfies the body of rule (3). Then I must satisfy its head, i.e., $I(b) \in [0.2, 0.3]$. However, $I(b) = 0.35 \notin [0.4, 0.5]$, and therefore rule (3) is not satisfied by I.■

We note that the fixpoint of P_1 is defined but it is not *tight* enough to represent exactly the set of satisfying p-interpretations. It is also possible for a p-program to have a well-defined fixpoint but be inconsistent. Consider p-program P_2 from Figure 1.

Proposition 2. *1.* $lfp(T_{P_2}) = T^1_{P_2}$. *In particular,* $lfp(T_{P_2})(a) = [0.2, 0.4]$ *and* $lfp(T_{P_2})(b) = [0.3, 0.5]$.
2. $Mod(P_2) = \emptyset$

Proof. The first part is similar to the proof of Proposition 1. To show that $Mod(P_2) = \emptyset$ consider some p-interpretation I such that $I \models P_2$. Let $I(a) = p$. As $p \in lfp(T_{P_2})(a) = [0.2, 0.4]$ then $p \in [0.2, 0.3]$, or $p \in [0.3, 0.4]$. In either case, the body of at least one of the rules (3),(4) will be satisfied by I and therefore, $I(b) \in [0.6, 0.7]$. However, we know that $I(b) \in lfp(T_{P_2})(b) = [0.3, 0.5]$, which leads to a contradiction. ■

Note that the $lfp(T_P)$ specifies the semantics of a p-program as the set of p-interpretations inside a single N-dimensional parallelepiped whose borders are defined by $lfp(T_P)(A_1), \ldots, lfp(T_P)(A_N)$. Unfortunately, this is not always the case, i.e., $Mod(P)$ need not be a single N-dimensional parallelepiped, as evidenced by the following proposition.

Proposition 3. *If the atoms in* B_L *for* P_1 *(Figure 1) are ordered as* a, b*, then* $Mod(P_3) = [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$.

Proof. First, we show that $Mod(P_1) \subseteq [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$.

Let $I \models P_1$. As $lfp(T_P(P_1))(A) = [0.2, 0.4]$ (by rule (1)), three cases are possible.

1. $I(A) \in [0.2, 0.3)$. Consider rules (3) and (4). As $I(A) \in [0.2, 0.3)$, the body of rule (3) will be true, and the body of rule (4) will be false. Thus, I must satisfy the head of (3), i.e., $I(B) \in [0.2, 0.3]$. Therefore $I \in [0.2, 0.3) \times [0.2, 0.3]$

2. $I(A) = 0.3$. In this case, the bodies of both rule (3) and rule (4) are satisfied, and therefore I must satisfy both heads of these rules, i.e., $I(B) \in [0.2, 0.3]$ and $I(B) \in [0.4, 0.5]$. But as $[0.2, 0.3] \cap [0.4, 0.5] = \emptyset$, we arrive to a contradiction. Therefore, for any p-interpretation $I \models P_1$, $I(A) \neq 0.3$.
3. $I(A) \in (0.3, 0.4]$. Here, the body of rule (3) will not be true, but the body of rule (4) will, therefore, I must satisfy the head of rule (4), i.e., $I(B) \in [0.4, 0.5]$. Then, $I \in (0.3, 0.4] \times [0.4, 0.5]$.

Combining the results of all three cases together we get $I \in [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$, which proves the inclusion. It is easy to verify that any $I \in [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$, is the model of P_1.■

We note, here, that, in general, the problem of determining if a simple p-program P is consistent is hard. We define the set CONS-P$= \{P | Mod(P) \neq \emptyset\}$.

Theorem 1. *The set CONS-P is NP-complete.*

3 Semantics of PLPs Revisited

As shown in Section 2.2, even for the simplest p-programs, the set of their models may have a more complex structure than the one prescribed by the fixpoint procedure of [9]. In this section we study the problem of exact description and explicit computation of $Mod(P)$ given program P. We show that in general case, $Mod(P)$ is a union of a finite number of N-dimensional[1] open, closed, or semi-closed parallelepipeds within the N-dimensional unit hypercube $[0, 1]^N$. In Section 3.1 we characterize $Mod(P)$ as the set of solutions of a family of systems of inequalities constructed from P. In Section 3.2 we propose a way of computing $Mod(P)$ using special transformations of P.

3.1 Characterization of Models

Definition 1. *Let P be a simple p-program over the Herbrand base $B_L = \{A_1, \ldots, A_N\}$. With each atom $A \in B_L$ we will associate a real variable x_A with domain [0,1]. Let $C \equiv A : [l, u] \longleftarrow B_1 : [l_1, u_1] \wedge \ldots \wedge B_k : [l_k, u_k]$, $k \geq 0$ be a clause of P.*

The family of systems of inequalities induced by C, *denoted $INEQ(C)$ is defined as follows:*

- $k = 0$ *(C is a fact).* $INEQ(C) = \{\{l \leq x_A \leq u\}\}$
- $k \geq 1$ *(C is a rule).*
  $INEQ(C) = T(C) \cup F(C)$;
  $T(C) = \{\{l \leq x_A \leq u, l_i \leq x_{B_i} \leq u_i | 1 \leq i \leq k\}\}$;
  $F(C) = \{\{x_{B_i} < l_i\} | 1 \leq i \leq k\} \cup \{\{x_{B_i} > u_i\} | 1 \leq i \leq k\}$.

[1] Whenever we are writing about N-dimensional parallelepipeds representing the set of models of a p-program, we implicitly assume the possibility that the true dimensionality of some of them can be less than N due to the fact that with certain atoms of B_L exact point probabilities, rather than intervals may be associated.

The family $INEQ(P)$ of systems of inequalities *is defined as* $INEQ(P) = \{\alpha_1 \cup \ldots \cup \alpha_m | \alpha_i \in INEQ(C_i), 1 \leq i \leq k\}$.

Note that all inequalities in all systems from the definition above involve only one variable. Given a system α of such inequalities, we denote the set of its solutions as $Sol(\alpha)$. For $A \in B_L$ let $l_A^\alpha = \max\{0 \cup \{l | (x_A \leq l) \in \alpha\}\}$ and $u_A^\alpha = \min\{1 \cup \{u | x_A \geq u \in \alpha\}\}$. Then it it easy to see that

$$Sol(\alpha) = \begin{cases} \emptyset & \text{if for some } \ A, \ l_A^\alpha > u_A^\alpha; \\ [l_{A_1}^\alpha, u_{A_1}^\alpha] \times \ldots \times [l_{A_N}^\alpha, u_{A_N}^\alpha] & \text{otherwise.} \end{cases}$$

Informally, the set $INEQ(P)$ represents all possible systems of restrictions on probabilities of atoms of B_L whose solutions satisfy every clause of P. Of course, not all individual systems of inequalities have solutions, but $INEQ(P)$ *captures all the systems that do*, as shown in the following lemma and theorem.

Lemma 1. *Let C be a p-clause and I be a $p - interpretation$ (both over the same Herbrand Base B_L). Then $I \models C$ iff $\{x_A = I(A)\} \in Sol(\alpha)$ for some $\alpha \in INEQ(C)$.*

Theorem 2. *A p-interpretation I is a model of a simple p-program P **iff** there exists a system of inequalities $\alpha \in INEQ(P)$ such that $X = \{x_A = I(A)\} \in Sol(\alpha)$.*

This leads to the following description of $Mod(P)$:

Corollary 1. $Mod(P) = \bigcup_{\alpha \in INEQ(P)} Sol(\alpha)$

We denote as $Facts(P)$ and $Rules(P)$ the sets of p-clauses with empty and non-empty bodies in a p-program P, and as $f(P)$ and $r(P)$ - their respective sizes. Let also $k(P)$ be the maximum number of atoms in a body of a rule in P. Then, we can obtain the following bound on the size of $Mod(P)$.

Corollary 2. *The set of all p-interpretations I that satisfy a simple p-program P is a union of at most $M(P)$ (not necessarily disjoint) N-dimensional parallelepipeds, where $M(P) = (2k(P) + 1)^{r(P)}$.*

This Corollary provides an exponential, in the size of the p-program, upper bound on the number of disjoint parallelepipeds in the set $Mod(P)$. We can show that this bound cannot be substantially decreased in the general case. Consider p-program P_3 over the set of atoms $\{a, b_1, \ldots, b_n\}$:
$a : [1,1] \longleftarrow .$ (1)
$b_i : [0,1] \longleftarrow . \ i = 1, \ldots, n$ (2i)
$a : [0,0] \longleftarrow b_i : [0.2, 0.3]. \ i = 1, \ldots, n$ (3i)

Here, $INEQ(1)$ consists of a single equality $x_a = 1$; each of $INEQ(2i)$ includes trivial inequalities $0 \leq x_{b_i} \leq 1$, and each of $INEQ(3i)$ consists of three systems of inequalities: $\alpha_i^1 = \{0 \leq x_{b_i} < 0.2\}$, $\alpha_i^2 = \{0.3 < x_{b_i} \leq 1\}$, and $\alpha_i^3 = \{0.2 \leq x_{b_i} < 0.3; x_a = 0\}$. Since α_i^3 is inconsistent with $INEQ(1)$, each consistent set of inequalities in $INEQ(P_3)$ can be represented as $\{x_a =$

$1\} \cup \bigcup_{i=1}^{n} \alpha_i^{j_i}$ for some $j_i \in \{1,2\}, i = 1,\ldots,n$. It is easy to see that for any two different α and α' of such form in $INEQ(P_3)$ sets $Sol(\alpha)$ and $Sol(\alpha')$ are disjoint. So, $Mod(P_3)$ consists of 2^n disjoint n-dimensional parallelepipeds. At the same time $f(P_3) = n+1,\ r(P_3) = n,\ k(P_3) = 1$ and a bitwise representation of P_3 takes only $O(n \log n)$ bits.

3.2 Explicit Computation of Models

In this section we will address the following problem: given a simple p-program P, output the description of the set $Mod(P)$ as a union of N-dimensional parallelepipeds.

The construction from previous section gives one algorithm for computing $Mod(P)$: given a program P construct explicitly the set of systems of inequalities $INEQ(P)$ and then solve each system from this set. This algorithm has exponential worst case complexity in the size of the program and as program P_3 illustrates the worst case cannot be avoided. However, it not hard to see that the algorithm based on solving individual systems of inequalities from $INEQ(P)$ can be quite inefficient in its work. Indeed, as the solution sets of individual systems of inequalities are not necessarily disjoint, this algorithm may wind up computing parts of the final solution over and over. In this section, we propose a different approach to direct computation of the set of models of a simple p-program, which breaks the solution space into *disjoint* components and individually computes each such component.

Consider a simple p-program P over the Herbrand base $B_L = \{A_1, \ldots A_N\}$. As $AT(P)$ we denote the multiset of all p-annotated atoms found in all heads and bodies of clauses in P. Given $A \in B_L$ Let $AT(P)[A]$ be the set of all p-annotated atoms of the form $A : \mu$ from $AT(P)$. Define for each $A \in B_L$ a set $Prb_P(A_i)$ of all possible bounds of probability intervals used in P for A_i as follows $Prb_P(A) = \{\langle l, -\rangle | A : [l,u] \in AT(P)[A]\} \cup \{\langle u, +\rangle | A : [l,u] \in AT(P)[A]\} \cup \{< 0, - >, < 1, + >\}$. Thus with each occurrence of a probability bound for A in P, we are also storing (encoded as "$-$" or "$+$") whether it is a lower or upper bound.

We order the elements of $Prb_P(A)$ as follows. $\langle a, *\rangle < \langle b, *\rangle$ whenever $a \leq b$, and $\langle a, -\rangle < \langle a, +\rangle$. Consider now $Prb_P(A) = \{\beta_1 =< 0, - >, \beta_2, \ldots, \beta_m =< 1, + >\}$ where sequence $\beta_1, \ldots, \beta_m$ is in ascending order. Using the set $Prb_P(A)$ we will now construct the set of segments $SEG_P(A)$ as follows.

Let $\beta_i = \langle a_i, \lambda_i\rangle$ and $\beta_{i+1} = \langle a_{i+1}, \lambda_{i+1}\rangle$, $1 \leq i \leq m-1$. We define the segment s_i associated with the pair β_i, β_{i+1} as shown in the table on the left side of Figure 2. Now, $SEG_P(A) = \{s_1, s_2, \ldots, s_{m-1}\}$.

Notice that if $a_i = a_{i+1}$ then, λ_i is a "$-$" and λ_{i+1} is a "$+$" (it follows from our order on β_is) and the interval $[a_i, a_{i+i}] = [a_i, a_i]$ will be added to $SEG_P(A)$. The following proposition establishes basic properties of the segment sets.

Proposition 4. *Let P be a simple p-program, $A \in B_L$ and $SEG_P(A) = \{s_1, \ldots, s_{m-1}\}$.*

1. *$SEG_P(A)$ is a partition of $[0,1]$, in particular,. if $i \neq j$ then $s_i \cap s_j = \emptyset$.*

λ_i	λ_{i+1}	s_i
$-$	$-$	$[a_i, a_{i+1})$
$-$	$+$	$[a_i, a_{i+1}]$
$+$	$-$	(a_i, a_{i+1})
$+$	$+$	$(a_i, a_{i+1}]$

```
(1) Compute $SEG(P)$.
(2) for each $J \in SEG(P)$ do
(3)     Choose some interpretation (point) $I \in J$;
(4)     if $I \models P$ then add $J$ to $Mod(P)$ end if
(5) end do
```

Fig. 2. Determination of segments in $SEG(A)$ (left) and algorithm GenMod for computing $Mod(P)$. (right)

2. *Consider some $1 \le i \le m-1$. Let x,$y \in s_i$ and let I_1 and I_2 be p-interpretations such that $I_1(A) = x$ and $I_2(A) = y$. Then for all $A : \mu \in AT(P)[A]$, $I_1 \models A : \mu$* ***iff*** *$I_2 \models A : \mu$.*
3. *Consider some $1 \le i \le m-2$. Let $x \in s_i$ and $y \in s_{i+1}$ and let I_1 and I_2 be p-interpretations such that $I_1(A) = x$ and $I_2(A) = y$. Then*

$$\{A : \mu \in AT(P)[A] \mid I_1 \models A : \mu\} \neq \{A : \mu \in AT(P)[A] \mid I_2 \models A : \mu\}.$$

Given a simple p-program P over the Herbrand base $B_L = \{A_1, \ldots, A_N\}$, the segmentation of P, denoted $SEG(P)$ is defined as follows

$$SEG(P) = \{s^1 \times s^2 \times \ldots \times s^N \mid s^j \in SEG_P(A_j), 1 \le j \le N\}.$$

Basically, $SEG(P)$ is a segmentation of the N-dimensional unit hypercube into a number of "bricks". Recall that each point inside the N-dimensional unit hypercube represents a p-interpretation. The following theorem shows that the set of all p-interpretations satisfying P can be constructed from some "bricks" of $SEG(P)$.

Theorem 3. *1. Any two different parallelepipeds of $SEG(P)$ do not intersect.*
2. For any parallelepiped $J \in SEG(P)$ either $J \subseteq Mod(P)$, or $J \cap Mod(P) = \emptyset$.
3. There exists such subset $S \subseteq SEG(P)$ that $Mod(P) = \bigcup_{J \in S} J$.

Consider again program P_1 (Fig. 1). Atom a has the set of probability bounds $Prb_{P_1}(a) = \{\langle 0, -\rangle, \langle 0.2, -\rangle, \langle 0.3, -\rangle, \langle 0.3, +\rangle, \langle 0.4, +\rangle, \langle 1, +\rangle\}$ and atom b has the set of bounds $Prb_{P_1}(b) = \{\langle 0, -\rangle, \langle 0.2, -\rangle, \langle 0.3, +\rangle, \langle 0.4, -\rangle, \langle 0.5, +\rangle, \langle 1, +\rangle\}$. The corresponding sets of the segments are
$SEG_{P_1}(a) = \{[0, 0.2], [0.2, 0.3), [0.3, 0.3], (0.3, 0.4], (0.4, 1]\}$ and
$SEG_{P_1}(b) = \{[0, 0.2), [0.2, 0.3], (0.3, 0.4), [0.4, 0.5], (0.5, 1]\}$.
Then $SEG(P_1)$ consists of 25 rectangles of the form $s^1 \times s^2$ where $s^1 \in SEG_{P_3}(a)$ and $s^2 \in SEG_{P_1}(b)$ (in fact, 5 of them with $s^1 = [0.3, 0.3]$ are linear segments). As is shown in Proposition 3 only 2 of them consist of models of P_3: $Mod(P_1) = [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$.

Theorem 3 suggests that $Mod(P)$ can be constructed using the algorithm *GenMod* described in Figure 2. We note that steps (3) and (4) of this algorithm can be processed efficiently. In particular, if $J = s^1 \times \ldots \times s^N$ and each s^i is a

```
Algorithm GenModT($P$:program, $\{A_1,\ldots,A_n\}$:atoms)
if $n=1$ and $P=\bigcup_{j=1}^{k}\{A_1:\mu_j\leftarrow .\}$ then $Sol:=\bigcap_{j=1}^{k}\mu_j$
else // $P$ includes at least two different atoms
  $Sol:=\emptyset$;
  $S:=NS(P)$;      // compute Ng-Subrahmanian $T_P$ operator
  if $S=\emptyset$ then return($\emptyset$)
  else // if NS(P) is not empty, proceed with computations
    // reduce $P$ wrt $S=\times_{i=1}^{n}sg_i$
    for $i=1$ to $n$ do   $P:=Reduct(P,A_i:sg_i)$   end_do
    $Seg:=SEG(P,A_1)\cap sg_1$;    // the segmentation of $A_1$ inside
                          // the $T_P$ operator
// main loop
    for each $s=\langle a,b\rangle\in Seg$ do
      $P':=Reduct(P,A_1:s)$;
      if $P'$ is empty then   $Sol:=Sol\cup(s\times(\times_{i=2}^{n}[0,1]))$
      else// find the solution for the reduct
        $RSol:=GenModT(P',\{A_2,\ldots,A_n\})$;
        if $RSol\neq\emptyset$ then   $Sol:=Sol\cup(s\times RSol)$  end if
      end if   end do
end if   end if
return $Sol$;
```

Fig. 3. Algorithm GenModT for computing $Mod(P)$.

segment with the lower bound l^i and the upper bound u^i, $i=1,\ldots,N$, then for each i the value $I(A_i)$ on step (3) can be chosen to be equal to $(l^i+u^i)/2$. So, the runtime of *GenMod* is bounded by a polynomial of the size of $SEG(P)$. The size of $SEG(P)$ is, in its turn, exponential of the size of the set B_L of all atoms of P. Of course, it can be a case when some “bricks” in $SEG(P)$ can be united into one larger “brick”, so that $Mod(P)$ is represented by a smaller number of bricks than $SEG(P)$. But the program P_3 shows that in the general case even minimal number of ”non-unitable” bricks in $Mod(P)$ can be exponential in $|B_L|$. Therefore, the worst case running time of algorithm *GenMod* can not be improved. At the same time, we can improve on *GenMod*, by being more careful at how the N-dimensional “bricks” are considered.

We fix an ordering $A_1,\ldots,A_N$ of B_L. Given a simple p-program P, let $lfp(T_P(A_i))=sg_i$ and $NS(P)=\times_{i=1}^{N}sg_i$. From [9] we know that $Mod(P)\in NS(P)$. We observe, that it is sufficient, to segment $NS(P)$ rather than the unit N-dimensional hypercube to compute $Mod(P)$. For a set of segments S and a segment μ let us denote by $S\cap\mu$ the set $\{s|s\in S$ and $s\subseteq\mu\}$.

Given a simple p-program P, an atom $A\in B_L$ and an interval $\nu\subseteq[0,1]$, we denote by $Reduct(P,A:\nu)$ a reduced program which results from P as follows:
(i) Delete from P any clause C with the head $A:\mu$ such that $\nu\subseteq\mu$.
(ii) Delete from P any clause C whose body includes an atom $A:\mu$ such that

$\mu \cap \nu = \emptyset$.
(ii) Delete from the body of any other rule each atom $A : \mu$ such that $\nu \subseteq \mu$.
It is easy to see that $Mod(Reduct(P, A : \nu)) = Mod(P \cup \{A : \nu \leftarrow .\})$.

Figure 3 contains the pseudocode for the algorithm GenModT, designed to intelligently execute all steps of the algorithm GenMod. The algorithm works as follows. On the first step, we compute $NS(P)$, reduce P wrt $NS(P)$ and construct segmentation of A_1. Then for each segment, we construct a reduced program P' and recursively run GenModT on P' and set $\{A_2, \ldots, A_n\}$ of atoms, and combine the solution returned by the recursive call with the segment of A_1 for which it was obtained. The union of solutions computed this way is returned at the end of each call to GenModT. The stopping conditions are either an empty reduct program, meaning that the segmentation leading to this reduct yields a part of the final solution, or a contradiction during the computation of $NS(P)$, meaning that current segmentation does not yield models of P. The theorem below states that Algorithm GenModT is correct.

Theorem 4. *Given a simple p-program P and an ordering $A_1, \ldots, A_N$ of B_L, algorithm GenModT returns the set $Mod(P)$.*

Apart from using $NS(.)$ as starting points for segmentation on every step, Algorithm GenModT improves over a naive implementation of GenMod in two ways. First, it may turn out that one of the stopping conditions for GenModT holds before the recursion has exhausted all atoms from P. In this case, it means that either an entire sub-space is part of the solution or is not part of the solution, but we no longer need to check each "brick" inside that sub-space. Second, on each step of the recursion after the first one, segmentation of the current atom occurs with respect to the current program, which is a reduct of P w.r.t. all previously considered atoms. This reduct has a simpler structure, and, in many cases, would have fewer and shorter rules. This means that the segmentation of the current atom w.r.t. the reduct may contain fewer segments than the segmentation w.r.t. original program P. Another convenient feature ofGenModT is that it structures $Mod(P)$ in a form of a tree, corresponding to the way it recursively enumerates the solutions.

The advantages of GenModT over naive implementation of GenMod are demonstrated in the example of program P_1 (Fig. 1). It was shown that $NS(P_1) = [0.2, 0.4] \times [0.2, 0.5]$ and that
$SEG_{P_1}(a) = \{[0, 0.2], [0.2, 0.3), [0.3, 0.3], (0.3, 0.4], (0.4, 1]\}$ and
$SEG_{P_1}(b) = \{[0, 0.2), [0.2, 0.3], (0.3, 0.4), [0.4, 0.5], (0.5, 1]\}$.
So, at the first step of GenModT $Seg = SEG_{P_1}(a) \cap (0.2, 0.4] = \{[0.2, 0.3), [0.3, 0.3], (0.3, 0.4]\}$ and the main loop will proceed three times as follows:
1) $s = [0.2, 0.3)$, $P' = \{b : [0.2, 0.3] \longleftarrow .\}$, $Sol = \{[0.2, 0.3) \times [0.2, 0.3]\}$;
2) $s = [0.3, 0.3]$, $P' = \{b : [0.2, 0.3] \longleftarrow .;\ b : [0.4, 0.5] \longleftarrow .\}$, $Sol := Sol \cup \emptyset$;
3) $s = (0.3, 0.4]$, $P' = \{\ b : [0.4, 0.5] \longleftarrow .\}$, $Sol := Sol \cup \{(0.3, 0.4] \times [0.4, 0.5]\}$.
The result will be $Sol = [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$ which is equal to $Mod(P_1)$ (see Proposition 3). Thus, GenModT tries only 3 bricks while GenMod will check all 25 bricks.

4 Related Work and Conclusions

There has been a number of logic programming frameworks for uncertainty proposed in the past 15 years (see [4] for a detailed survey), most concentrating on point probabilities. The work of Poole [13] and Ngo and Haddawy [12] treated the "$\longleftarrow$" as conditional dependence and used logic programming to model Bayesian Networks. In more recent work, Baral et al.[1] present an elegant way to incorporate probabilistic reasoning into an answer set programming framework, in which they combine probabilistic reasoning with traditional non-monotonic reasoning. At the same time,some work [8–11, 4, 5] looked at interval probabilities as the means of expressing imprecision in probability assessment. tics of the original In all those frameworks, the underlying semantics allowed for expression of the possible probability of an atom in a program as a single closed interval. Our work is the first to consider a harder problem of describing the semantics of interval-based probabilistic logic programs with sets of point probability assessments (p-interpretations), based on the semantics of interval probabilities proposed by De Campos et. al [3] and Weichselberger[15]. As shown in this paper, even for fairly simple syntax, such descriptions become more complex than single intervals and their computation is much more strenuous. Our next step is to study our semantics in the full language of p-programs of [9] and hybrid probabilistic programs [4]. We are also interested in investigating the relationship between the p-programs with possible worlds semantics and constraint logic programs.

References

1. Chitta Baral, Michael Gelfond, J. Nelson Rushton. (2004) Probabilistic Reasoning With Answer Sets, in *Proc. LPNMR-2004*, pp. 21-33.
2. V. Biazzo, A. Gilio. (1999) A Generalization of the Fundamental Theorem of de Finetti for Imprecise Conditional Probability Assessments, *Proc. 1st. Intl. Symposium on Imprecise Probabilities and Their Applications.*
3. Luis M. de Campos, Juan F. Huete, Serafin Moral (1994). Probability Intervals: A Tool for Uncertain Reasoning, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (IJUFKS)*, Vol. 2(2), pp. 167 – 196.
4. A. Dekhtyar and V.S. Subrahmanian (2000) Hybrid Probabilistic Programs. *Journal of Logic Programming*, Volume 43, Issue 3, pp. 187 – 250 .
5. M.I.. Dekhtyar, A. Dekhtyar and V.S. Subrahmanian (1999) Hybrid Probabilistic Programs: Algorithms and Complexity in Proc. of 1999 Conf. on Uncertainty in AI (UAI), pp 160 – 169.
6. H.E. Kyburg Jr. (1998) Interval-valued Probabilities, in *G. de Cooman, P. Walley and F.G. Cozman (Eds.), Imprecise Probabilities Project*, http://ippserv.rug.ac.be/documentation/interval_prob/interval_prob.html.
7. V.S. Lakshmanan and F. Sadri. (1994) *Modeling Uncertainty in Deductive Databases*, Proc. Int. Conf. on Database Expert Systems and Applications, (DEXA'94), September 7-9, 1994, Athens, Greece, Lecture Notes in Computer Science, Vol. 856, Springer (1994), pp. 724-733.
8. V.S. Lakshmanan and F. Sadri. (1994) *Probabilistic Deductive Databases*, Proc. Int. Logic Programming Symp., (ILPS'94), November 1994, Ithaca, NY, MIT Press.

9. R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, INFORMATION AND COMPUTATION, 101, 2, pps 150–201, 1993.
10. R. Ng and V.S. Subrahmanian. A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases, JOURNAL OF AUTOMATED REASONING, 10, 2, pps 191–235, 1993.
11. R. Ng and V.S. Subrahmanian. (1995) *Stable Semantics for Probabilistic Deductive Databases*, INFORMATION AND COMPUTATION, 110, 1, pps 42-83.
12. L. Ngo, P. Haddawy (1995) Probabilistic Logic Programming and Bayesian Networks, in *Proc. ASIAN-1995*, pp. 286-300.
13. D. Poole (1993). Probabilistic Horn Abduction and Bayesian Networks. *Artificial Intelligence*, Vol. 64(1), pp. 81-129.
14. Walley, P. (1991). Statistical Reasoning with Imprecise Probabilities. Chapman and Hall, 1991.
15. Weichselberger, K. (1999). The theory of interval-probability as a unifying concept for uncertainty. *Proc. 1st International Symp. on Imprecise Probabilities and Their Applications.*