Noname manuscript No.
(will be inserted by the editor)

The Theory of Interval Probabilistic Logic Programs

Alex Dekhtyar[1]*, **Michael I. Dekhtyar**[2]**

[1] Department of Computer Science, Cal Poly, San Luis Obispo, CA, e-mail: `dekhtyar@csc.calpoly.edu`
[2] Department of Computer Science, Tver State University, Tver, Russia, e-mail: `Michael.Dekhtyar@tversu.ru`

The date of receipt and acceptance will be inserted by the editor

Abstract Two approaches to logic programming with probabilities emerged over time: Bayesian reasoning and probabilistic satisfiability (PSAT). The attractiveness of the former is in tying the logic programming research to the body of work on Bayes networks. The second approach ties, from the point of view of computation, reasoning about probabilities to linear programming, and allows for natural expression of imprecision in probabilities via the use of intervals.

In this paper we construct precise semantics for one PSAT-based formalism for reasoning with interval probabilities: disjunctive probabilistic logic programs (dp-programs). It has two origins: (1) disjunctive logic programs, a powerful language for knowledge representation, first proposed by Minker in the early eighties [20] and (2) a logic programming language with interval probabilities originally considered by Ng and Subrahmanian [21, 22]. We show that the probability ranges of atoms and formulas in dp-programs cannot be expressed as single intervals. We construct the precise description of the set of models for the class of dp-programs and study the computational complexity of this problem, as well as the problem of consistency of a dp-program. We also study the conditions under which our semantics coincides with the single-interval semantics originally proposed by Ng and Subrahmanian. Our work sheds light on the complexity of constructing reasoning formalisms for imprecise probabilities and suggests that interval probabilities alone are inadequate to support such reasoning.

* Work partially supported by NSF grant ITR-0325063.
** Work partially supported by Russian Foundation for Basic Research grant 07-01-00637.

Key words probabilistic logic programming – interval probabilities – probabilistic satisfaction – possible worlds semantics

1 Introduction

Reasoning with probabilistic information, in the context of logic programming, has two distinct origins: Bayesian reasoning and probabilistic satisfiability. Logic programming frameworks based on the Bayesian reasoning paradigm interpret statements about conditional probability of event A: $P(A|B) = p$, as an implication of a special kind,

$$A : p \longleftarrow B.$$

This implication is usually read as follows: `"if` B `then the probability of` A `is equal to` p`"`. Among the logic programming frameworks following this idea are the work of Poole[27], Ngo and Haddawy [24], and more recently, and in the context of answer set programming, of Baral, Gelfond and Rushton [2].

Probabilistic Satisfiability (PSAT) inspires the alternative approaches to probabilistic logic programming frameworks. PSAT traces its origins to the work of Boole on probability theory [1]. In his work, Boole considered a finite collection of propositional (boolean) formulas $F_1, \ldots, F_m$ over some finite collection of propopositions $A_1, \ldots A_n$. He then assumed that instead of knowing whether each formula is true, he is given a set of probability assignments to these formulas of the form

$$Prob(F_i) = p_i, 1 \leq i \leq m.$$

The question Boole had asked was whether it is possible to assign probabilities $Prob(A_j) = q_j, 1 \leq j \leq n$ to propositional atoms so that all the statements about the probabilities of $F_1, \ldots, F_m$ were true[1].

Unlike his work in logic, Boole's work on probability theory has been criticized by his colleagues and had not achieved the same prominence. However the PSAT problem, as formulated by Boole has been resurrected by Hailperin [12] more than a century later and has been "modernized" by Georgakopoulos, Kavvadis and Papadimitriou [11] in 1988. Nilsson's probabilistic logic [25] is based on PSAT and uses the semantics of possible worlds (world probability functions) to model probabilities of events. In [11] it is shown that PSAT is NP-complete.

The attractiveness of building logic programming frameworks based on Bayesian reasoning lies in direct relationship to the large body of work on

[1] An interesting historical note is that to solve this problem Boole has discovered linear programming or at least, a special case of the general problem. He suggested, albeit without proof, that solutions to his system should be sought at the extreme points of the region described by the system — almost inventing the simplex algorithm [1].

Bayesian networks and Markov Decision Processes. The attractiveness of PSAT-based logic programs is in the fact that PSAT has a natural extension to the case of imprecise probabilities. The importance of imprecise probabilities has been observed by numerous researchers in the past two decades [29,3]. It has led to the establishment of the Imprecise Probabilities Project [14] and has generated an interest in interval probabilistic logic programming frameworks [21,8,16].

Interval PSAT is a reformulation of PSAT, in which probability assignments of the form $P(F) = p_F$ are replaced with inequalities of the form $l_F \leq P(F) \leq u_F$. The underlying semantics and the methodology for solving Interval PSAT is the same as for PSAT. Logic programming frameworks inspired by PSAT consider rules of the form "$P(F) = \mu$ if $P(F_1) = \mu_1$ and $\dots$ and $P(F_n) = \mu_n$". Unlike in bayesian-inspired frameworks, here "if" is the classical logical implication. Logic programming formalisms stemming from PSAT, in which probabilities of events are expressed as intervals, have been considered by Ng and Subrahmanian[21,22] and by Dekhtyar and Subrahmanian[8]. In these frameworks, the fixpoint semantics of formulas, i.e., the set of possible probability assignments for them, had been represented using a single interval.

Our more recent work [6,7] extended the semantics of interval probabilistic logic programming languages based on Interval PSAT, in particular, the languages considered by Ng and Subrahmanian in [21,22]. When possible world semantics of Nilsson [25] is used, it turns out that the set of possible probability values for a formula in a probabilistic program may no longer be a single closed interval - and thus, the previously used fixpoint procedures are no longer adequate for explicit description of the set of models of the probabilistic programs. In [6] we have described the new semantics for the class of probabilistic programs that involved only atomic formulas in the heads of clauses. We provided the exact mathematical description of the set of models for an interval probabilistic program and also, have presented procedures for building this set explicitly. In [7] we considered the semantics of a broader class of programs, equivalent to those studied by Ng and Subrahmanian in [22][2].

This paper extends our prior results to a larger class of programs: disjunctive interval probabilistic programs (dp-programs). These programs, based on disjunctive logic programs introduced by Minker [20] are interval probabilistic programs where heads of rules can contain disjunctions of formulas. An example of a rule in our logic programming language is given below:

$$a : [0.3, 0.4] \vee (b \wedge c) : [0.4, 0.6] \longleftarrow d : [0.3, 0.5], (a \vee f) : [0.3, 0.6].$$

This rule is interpreted (rather verbosely) as follows: "If the probability of event d is between 30% and 50% and the probability that at least one of a or f holds is between 30% and 60% then conclude that at least one of the

[2] Without variable annotations for probabilities.

following is true: the probability of a is between 30% and 40% or the probability of both b and c happening together is between 40% and 60%.".

The main contributions of the paper are:

- *Mathematical description of the set of models* for dp-programs.
- *Study of consistency problem* for dp-programs. We show that deciding whether a dp-program has a model is NP-complete.
- *Procedure for explicit model construction* for a subclass of dp-programs. We show that for a significant subclass of dp-programs, it is possible to construct the description of the set of models explicitly and in the form of a union of disjoint subsets.
- *Conditions for existence of a simple set of models.* Under some conditions the set of models for a dp-program can coincide with the result of the fixpoint procedure of [21,22][3]. We present some such conditions.

The rest of the paper is organized as follows. Section 2 introduces the syntax of dp-programs, their model-theoretic semantics (possible worlds probability density functions and probabilistic interpretations) and studies the problem of consistency of dp-programs, i.e., the problem of determining if a given program has a model. Section 3 describes the fixpoint procedure for dp-programs and shows the circumstances under which the fixpoint procedure does not yield the exact description of the set of models for a dp-program. In Section 4 we provide the exact characterization of the set of models for the class of dp-programs. We also discuss the simplified characterization for a subclass called *simple* dp-programs: programs with only atomic formulas in the heads of clauses. In Section 5 we describe conditions which allow dp-programs to have semantics that coincides with the fixpoint semantics of [22].

2 Interval Probabilistic Logic Programs

2.1 Syntax

In this section we describe the syntax of the interval probabilistic logic programming language that we study in this paper. Our earlier work [6,7] used the syntax of probabilistic logic programs of Ng and Subrahmanian [21,22] with constant probability annotations. In present work, we extend the class of programs we study to the case of disjunctive logic programs.

Let $B_L = \{A_1, \ldots, A_N\}$ be a finite set of propositional atoms[4]. A *basic formula* is either an atom from B_L or a conjunction or disjunction of two or more atoms. The set of all basic formulas is denoted $bf(B_L)$. Formulas of the form $(B_1 \wedge \ldots \wedge B_n) : \mu$ and $(B'_1 \vee \ldots \vee B'_m) : \mu'$, where $B_1, \ldots, B_n, B'_1, \ldots, B'_m \in B_L$ and $\mu = [l, u], \mu' = [l', u'] \subseteq [0, 1]$ are called *p-annotated conjunctions* and *p-annotated disjunctions* respectively.

[3] Extended naturally to incorporate disjunctive rules.

[4] We use B_L to be consistent in our notation with [21].

P-annotated conjunctions and disjunctions represent probabilistic information. Every atom in B_L is assumed to represent an (uncertain) event or statement. A *p-annotated conjunction* $A_1 \wedge \ldots \wedge A_n : [l, u]$ is read as "the probability of the joint occurrence of the events corresponding to $A_1, \ldots, A_n$ lies in the interval $[l, u]$". Similarly, $A_1 \vee \ldots \vee A_n : [l, u]$ is read as "the probability of the occurrence of at least one of the events corresponding to $A_1, \ldots, A_n$ lies in the interval $[l, u]$".

Disjunctive Probabilistic Logic Programs (dp-programs) are constructed from *p-annotated* formulas as follows.

Definition 1 *Let* $G_1, \ldots, G_k, F_1, \ldots, F_n$ *be some basic formulas and* $\nu_1, \ldots, \nu_k, \mu_1, \ldots, \mu_n$ *be closed subintervals of* $[0, 1]$ *(also called* annotations*). Then, a disjunctive p-clause is an expression of the form*

$$G_1 : \nu_1 \vee \ldots \vee G_k : \nu_k \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n.$$

where $k \geq 0$ *and* $n \geq 0$.

The head *of the clause is its left side* $G_1 : \nu_1 \vee \ldots \vee G_k : \nu_k$ *and* the body *of the clause is its right side* $F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$.

If $k = 1$*, the p-clause* $G_1 : \nu_1 \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ *is non-disjunctive. Our earlier work [7] considered only such clauses, as did [21, 22].*

If $k > 0, n = 0$*, as usual, the p-clause* $G_1 : \nu_1 \vee \ldots \vee G_k : \nu_k \longleftarrow .$ *is referred to as a* fact.

If $k = 0, n > 0$*, as usual, the p-clause* $\longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n.$ *is referred to as an* integrity constraint.

If $k = 0$ *and* $n = 0$ *then empty clause* $\longleftarrow .$ *denotes a contradiction.*

Definition 2 *A* Disjunctive Probabilistic Logic Program *(*dp-program*) is a finite collection of* disjunctive p-clauses.

A dp-program in which all clauses are non-disjunctive is called a non-disjunctive p-program *or simply a* p-program.

A (d)p-program in which all basic formulas in all clauses are atoms from B_L *only is called* a simple (d)p-program.

A (d)p-program in which the heads of all clauses are atoms or disjunctions of atoms from B_L *is called a* factored (d)p-program.

Example 1 Consider a dp-program P consisting of the following p-clauses (defined over the Herbrand base $\{smoke, hot, dry, alarm, robbery, intrusion, fire\}$):

$smoke : [0.2, 0.4]$	$\longleftarrow$	.
	$\longleftarrow$	$smoke : [1, 1] \wedge (hot \wedge dry) : [0, 0].$
$alarm : [0.3, 0.6]$	$\longleftarrow$	$smoke : [0.6, 1] \wedge (hot \wedge dry) : [0.4, 0.7].$
$alarm : [0.7, 0.9]$	$\longleftarrow$	$smoke : [0.6, 1] \wedge (hot \wedge dry) : [0.8, 1].$
$fire : [0.4, 1] \vee robbery : [0.1, 0.2]$	$\longleftarrow$	$alarm : [0.4, 0.7], smoke : [0.8, 1].$
$(robbery \vee intrusion) : [0.6, 1]$ $\vee fire : [0.1, 0.3]$	$\longleftarrow$	$alarm : [0.8, 1] \wedge smoke : [0, 0.3] \wedge$ $(hot \wedge dry) : [0.2, 0.7].$

P describes a portion of a knowledge base about the work of a home alarm system. The first p-clause of P is a states that at present time the probability of smoke on the premises guarded by the alarm system is between 20% and 40%. The second p-clause is an integrity constraint that states, that confirmed smoke (i.e., smoke with probability of 100%) cannot happen when there is no chance (probability of 0%) for hot and dry weather. The third and fourth clauses specify the probability of an alarm given different probability estimates for the hot and dry weather when the probability of smoke is judged to be above 60%. The alarm has a higher chance of being triggered if the probability of hot and dry weather is in a higher range. The last two clauses contain disjunctions in the heads. These clauses identify the reasons for the alarm under different conditions. The first of the two clauses states that in the situation when the probability of an alarm is between 40% and 70% while the probability of smoke is judged to be above 80%, there is over 40% chance that there is a fire or a 10% to 20% chance that a robbery is in progress. The second clause states that if the probability of the alarm is judged to be above 80%, while the probability of smoke is less than 30% and the probability that the weather is hot and dry is between 20% and 70%, we have to conclude that there is a high — over 60% chance that we are dealing with either a robbery or an intrusion, or that there is a 10% to 30% chance that there is a fire. We note that the disjuncts in the heads of the p-clauses are not disjoint.

A p-program P whose clauses are all facts can be viewed as an instance of interval PSAT (although, due to the specificity of formulas in B_L, not every interval PSAT instance can be described this way). Notice, however, that while Horn clause logic programs are instances of SAT, the same is not true for p-programs. A (d)p-program containing non-trivial rules, in general, is not an instance of interval PSAT, as a generic (d)p-clause cannot be reduced to a p-annotated conjunction or disjunction. Below we show that a dp-program can be represented by a (possibly large) collection of interval PSAT instances.

In [21] Ng and Subrahmanian considered factored p-programs. In [23] they considered a framework, in which variables were allowed in the probability annotations. An example of a variable annotation is a rule:

$$alarm : [\alpha, \max(0.9 \cdot \beta, \alpha)] \longleftarrow smoke : [\alpha, \beta] \wedge (hot \wedge dry) : [0.7, 1].$$

This rule states, that if the probability of the weather being hot and dry is above 70%, then the alarm will sound with the probability in the interval that depends on the probability interval of smoke. If the probability interval for smoke is, for example, $[0.4, 0.5]$, then the alarm will sound with probability $[0.4, \max(0.9 * 0.5, 0.4)] = [0.4, 0.45]$.

Our definition of dp-programs introduces disjunctions in the heads and allows arbitrary heads of p-clauses, but does not consider variable annotations for probabilities.

2.2 Model Theory

The model theory assumes that in the real world each atom from B_L, and therefore each basic formula, is either true or false. However, exact information about the real world is not known. The uncertainty about the world is represented in a form of a probability distribution over the set of 2^N possible worlds. In addition, p-programs introduce uncertainty about the probability distribution itself.

More formally, the model theory is defined as follows.

Definition 3 *Let B_L be the Herbrand base of the language L described in Section 2.1. A world probability density function WP is defined as*
$WP : 2^{B_L} \to [0,1]$,
$\sum_{W \subseteq B_L} WP(W) = 1$.

Each subset W of B_L is considered to be a *possible world* and WP associates a point probability with it. Next we define the satisfaction relation on basic formulas.

Definition 4 *Let $A, A_1, \ldots A_n \in B_L$, and let $W \subseteq B_L$ be a possible world.*
$W \models A$ *iff* $A \in W$;

$W \models A_1 \wedge \ldots \wedge A_n$ *iff* $(\forall 1 \leq i \leq n) W \models A_i$;

$W \models A_1 \vee \ldots \vee A_n$ *iff* $(\exists 1 \leq i \leq n) W \models A_i$.

To simplify notation, in the remainder of the paper we fix an enumeration $W_1, \ldots W_M$, $M = 2^N$ of the possible worlds and denote $WP(W_i)$ as p_i.

World probability density functions represent the "base" of the possible worlds semantics. However, due to their size and granularity, they are not convenient to reason with. Below we introduce *atomic probabilistic interpretations (ap-interpretations)* and *probabilistic interpretations (p-interpretations)*: functions whose domains are atoms or basic formulas respectively, rather than possible worlds. Given a formula and a world probability density function, the probability of the formula is established in a straighforward way: it is the sum of probabilities on all possible worlds that satisfy the formula.

Definition 5 *An* atomic probabilistic interpretation *(*ap-interpretation*)* I^a *is a function* $I^a : B_L \to [0,1]$.

A probabilistic interpretation *(*p-interpretation*)* I *is a function* $I : bf(B_L) \to [0,1]$.

Definition 6 *Let WP be a world probability density function. A* p-interpretation I_{WP} *is defined as follows:* $I_{WP}(F) = \sum_{W \models F} WP(W)$.

Note that each world probability density function WP has a unique p-interpretation I_{WP} associated with it. However, in general, a p-interpretation

I can be induced by more than one world probability density function. Also, some p-interpretations can have no world probability density functions associated with them. We call the latter p-interpretations *inconsistent*, and, in general, do not consider them in this paper.

Example 2 Consider the Herbrand universe $B_L = \{a, b\}$. Let the world probability density function WP be defined as follows:

$WP(\emptyset) = p_1 = 0.2 \qquad WP(\{a, b\}) = p_4 = 0.3$

$WP(\{a\}) = p_2 = 0.4 \quad WP(\{b\}) = p_3 = 0.1$

Then, WP induces the following p-interpretation $I = I_{WP}$:

$I(a) = p_2 + p_4 = 0.7$;

$I(b) = p_3 + p_4 = 0.4$;

$I(a \wedge b) = p_4 = 0.3$;

$I(a \vee b) = p_2 + p_3 + p_4 = 0.8$

Example 3 Consider the following p-interpretation I: $I(a) = 1$, $I(b) = 1$, $I(a \wedge b) = 0$, $I(a \vee b) = 1$. We can show that this p-interpretation is inconsistent, i.e., there is no world probability density function WP corresponding to I. Using the notation from the previous example, we can form the following equations:

$I(a) = p_2 + p_4 = 1$;

$I(b) = p_3 + p_4 = 1$;

$I(a \wedge b) = p_4 = 0$;

$I(a \vee b) = p_2 + p_3 + p_4 = 1$,

to which we must add the $p_1 + p_2 + p_3 + p_4 = 1$ equation, which is part of the definition of a world probability density function and $p_i \geq 0$ inequalities for $i = 1, 2, 3, 4$, which specify that p_is are probabilities, and thus are non-negative.

The third equation above, $p_4 = 0$, entails immediately (substituting into the first and the second equations), that $p_2 = 1$ and $p_3 = 1$, which in turn means that $p_1 + p_2 + p_3 + p_4 > 1$, i.e., the above system of equations and inequalities does not have a solution. Thus, no world probability density function corresponding to the given p-interpretation can be found.

Given a p-interpretation I the corresponting ap-interpretation I^a is obtained by restricting the domain of I to B_L. In the other direction, given an ap-interpretation I^a, there is a nonempty set of p-interpretations $I : bf(B_L) \to [0, 1]$ such that $I(A) = I^a(A)$ for each atom $A \in B_L$.

While Example 3 shows that not every p-interpretation has a corresponding world probability density function, it is, in fact, the case that every ap-interpretation has one.

Proposition 1 *For each ap-interpretation I^a there is a world probability density function WP such that $I_{WP}(A) = I^a(A)$ for each atom $A \in B_L$.*

We can now define the model-theoretic semantics of dp-programs.

Definition 7 *Let $G_1, \ldots, G_k, F_1, \ldots, F_n$ be some basic formulas and $\nu_1, \ldots, \nu_k, \mu_1, \ldots, \mu_n$ be closed subintervals of $[0,1]$, and let I be a p-interpretation over $bf(B_L)$.*

- $I \models F_1 : \mu_1$ ***iff*** $I(F_1) \in \mu_1$;
- $I \models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ ***iff*** $(\forall 1 \leq i \leq n)(I \models F_i : \mu_i)$;
- $I \models G_1 : \nu_1 \vee \ldots \vee G_k : \nu_k$ ***iff*** $(\exists 1 \leq i \leq k)(I \models G_i : \nu_i)$;
- $I \models G_1 : \nu_1 \vee \ldots \vee G_k : \nu_k \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ ***iff*** *either* $I \models G_1 : \nu_1 \vee \ldots \vee G_k : \nu_k$ *or* $I \not\models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$.
  In particular,
  $I \models \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ ***iff*** $I \not\models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$,
  and always $I \not\models \longleftarrow$.

Definition 8 *Given a dp-program P, and a p-interpretation I, $I \models P$ (I is a model of P) iff there exists a world probability function WP, such that $I = I_{WP}$, and for all p-clauses $C \in P$, $I \models C$.*

Given a simple *dp-program P and an ap-interpretation I^a, $I^a \models P$ iff for all p-clauses $C \in P$, $I^a \models C$.*

Definition 9 *Let P be a dp-program. As $Mod(P)$ we denote the set of all models of P. This set is uniquely defined by the corresponding set of world probability functions*

$$WMod(P) = \{WP \mid I_{WP} = I \text{ for some } I \in Mod(P)\}.$$

We refer to a function WP from $WMod(P)$ as a w-model *of P and write $WP \models_w P$.*

P is called consistent iff $Mod(P) \neq \emptyset$ (or, which is equivalent, iff $WMod(P) = \emptyset$), otherwise P is called inconsistent.

It is convenient to view a single world probability function WP as a point $(WP(W_1), \ldots, WP(W_M))$ in the $M = 2^N$-dimensional unit cube E^M. Then, $WMod(P)$ can be viewed as a subset of E^M.

2.3 Consistency of dp-programs

In this section we consider briefly the problem of *consistency*, i.e., of existence of a model for the class of dp-programs. The consistency problem for dp-programs is defined as follows: given a dp-program P, check whether P has a model, i. e. $Mod(P) \neq \emptyset$ or, similarly, $WMod(P) \neq \emptyset$. Let CONS-P$= \{P | Mod(P) \neq \emptyset\}$.

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

Proof. *Upper bound.* Let P be a dp-program, $B_1, \ldots, B_r$ be all basic formulas of P Then $WMod(P) \neq \emptyset$ iff there exist such probabilities $b_1, \ldots, b_r$ of $B_1, \ldots, B_r$ that the system of linear equations and inequalities $EQ(P)$:

- $\sum_{W_j \models B_i} p_j = b_i$, for $i = 1, \ldots, r$,
- $\sum_{j=1}^{M} p_j = 1$;
- $p_j \geq 0$, for all $1 \leq j \leq M$.

has a solution $WP = \{p'_1 \ldots, p'_M\} \in WMod(P)$.

In general, WP is exponential in the size of the dp-program P, and therefore, we cannot simply guess any solution WP of $EQ(P)$. However, it turns out that if a good solution exists, then, there is also a good solution of polynomial size. In particular, we use the following lemma from [9] (which, in turn, cites [4]. A similar statement is also found in [11]).

Lemma 1 *If a system of r linear equations and/or inequalities with integer coefficients each of length at most l has a nonnegative solution, then it has a nonnegative solution with at most r entries positive, and where the size of each member of the solution is $O(rl + r\log r)$.*

Based on this lemma we obtain the following "small model" theorem.

Lemma 2 *A dp-program P including r different basic formulas is consistent iff there exists a probability distribution WP on possible worlds with no more than $r + 1$ nonzero probabilities such that $WP \models_w P$.*

Let the longest number in annotations of P have length l. Then the following nondeterministic procedure allows us to check whether $WMod(P) \neq \emptyset$.

1) Guess for each $B_i (i = 1, \ldots, r)$ its probability $b_i \in [0, 1]$ of the length $O(rl + r\log r)$.
2) Guess a probability distribution WP with no more than $r + 1$ positive probabilities $p_{i_1}, \ldots, p_{i_{r+1}}$ of the length $O(rl + r\log r)$ and check that WP is a solution of the system $EQ(P)$.
3) If $WP \models_w P$ return "Yes".

From the lemmas above it follows that this algorithm runs in nondeterministic time bounded by a polynomial of $|P|$.

Lower bound. We prove the lower bound for a subclass of simple p-programs with clause bodies of size 3 or less. We show that 3-CNF $\leq_P$ CONS-P. Let $\Phi = C_1 \wedge \ldots C_m$ be a 3-CNF over the set of boolean variables $Var = \{x_1, \ldots, x_n\}$. Let each clause $C_j, j = 1, \ldots, m$, include 3 literals l_j^1, l_j^2, l_j^3. Define for each literal l an annotated atom $\alpha(l)$ as follows: if $l = x \in Var$ then $\alpha(l) = x : [0.5, 1]$, if $l = \neg x$ then $\alpha(l) = x : [0, 0.5]$. Let $B_L = Var \cup \{C_j |\ j = 1, \ldots, m\} \cup \{\Phi\}$. We include in p-program $P(\Phi)$ the following p-clauses. $(f1): \quad \Phi : [1, 1] \leftarrow .$
$(fc_j): \quad C_j : [0, 0.1] \leftarrow . \quad (j = 1, \ldots, m)$
$(fx_i): \quad x_i : [0, 1] \leftarrow . \quad (i = 1, \ldots, n)$
$(rc_j): \quad C_j : [0.9, 1] \leftarrow \alpha(l_j^1) \wedge \alpha(l_j^2) \wedge \alpha(l_j^3). \quad (j = 1, \ldots, m).$
$(rf_i): \quad \Phi : [0, 0] \leftarrow x_i : [0.5, 0.5]. \quad (i = 1, \ldots, n)$

It is easy to see that $P(\Phi)$ can be constructed from Φ in polynomial time. Now the theorem follows from the following proposition.

Proposition 2 *$\Phi \in$ 3-CNF $\iff P(\Phi) \in$ CONS-P.*

Proof. Suppose that $\Phi \in$ 3-CNF and $\sigma : Var \to \{T, F\}$ is a truth substitution such that $\sigma(\Phi) = T$. Then for each $j = 1, \ldots, m$ there is such k_j, $1 \leq k_j \leq 3$, that $\sigma(l_j^{k_j}) = T$. Define an ap-interpretation I^a as follows: $I^a(\Phi) = 1, I^a(C_j) = 0$ for $j = 1, \ldots, m$, $I(x_i) = 0$ if $\sigma(x_i) = T$ and $I^a(x_i) = 1$ if $\sigma(x_i) = F$, $i = 1, \ldots, n$. Then it is easy to see that all facts $(f1), (fc_j), (fx_i)$ and all rules (rf_i) are valid on I^a. Consider now any rule (rc_j). Its body includes the annotated atom $\alpha(l_j^{k_j})$. If $l_j^{k_j} = x \in Var$ then $\alpha(l_j^{k_j}) = x : [0.5, 1]$. By the choice of $l_j^{k_j}$ we have that $\sigma(x) = T$, $I(x) = 0$ and therefore $I^a \not\models \alpha(l_j^{k_j})$. If $l_j^{k_j} = \neg x$ then $\alpha(l_j^{k_j}) = x : [0, 0.5]$. Again, by the choice of $l_j^{k_j}$ we have that $\sigma(x) = F$ and $I^a(x) = 1$ and therefore $I^a \not\models \alpha(l_j^{k_j})$. We see that in the both cases $I^a \not\models Body(rc_j)$ and hence, $I^a \models (rc_j)$. Therefore, $I^a \models P(\Phi)$.

Now suppose that there is model I^a of $P(\Phi)$. Then fact $(f1)$ implies $I^a(\Phi) = 1$, and it follows due to rules (rf_i) that $I(x_i) \neq 0.5$ for $i = 1, \ldots, n$. For $x \in Var$ we define $\sigma(x) = T$ if $I(x) < 0.5$ and $\sigma(x) = F$ if $I^a(x) > 0.5$. Show now that each clause C_j, $j = 1, \ldots, m$, includes such literal $l_j^{k_j}$ that $\sigma(l_j^{k_j}) = T$. Let us fix any j. The fact (fc_j) implies that inequality $I^a(C_j) \leq 0.1$ holds. Then the head $C_j : [0.9, 1]$ of the rule (rc_j) is not valid on I^a. Hence, there is an annotated atom in the body of (rc_j) which does not hold on I^a. Let it be atom $\alpha(l_j^{k_j})$. If $l_j^{k_j} = x$ for some $x \in Var$ then $\alpha(l_j^{k_j}) = x : [0.5, 1]$. Since $I^a \not\models x : [0.5, 1]$, we get that $I^a(x) < 0.5$ and $\sigma(l_j^{k_j}) = \sigma(x) = T$. If $l_j^{k_j} = \neg x$ for some $x \in Var$ then $\alpha(l_j^{k_j}) = x : [0, 0.5]$. Since $I^a \not\models x : [0, 0.5]$, we get that $I(x) > 0.5$. Then $\sigma(x) = F$ and $\sigma(l_j^{k_j}) = \neg\sigma(x) = T$. ■

Definition 10 *A consistent dp-program P* entails *a formula $F : [l, u]$ if for each $I \in Mod(P)$ $I \models F : [l, u]$. The entailment problem is, thus, expressed as follows: given a consistent P and a formula $F : [l, u]$, decide if P entails $F : [l, u]$.*

Let $EQ_1(P, F) = EQ(P) \cup \{\sum_{W_j \models F} p_j < l\}$ and $EQ_2(P, F) = EQ(P) \cup \{\sum_{W_j \models F} p_j > u\}$. Here, *$P$ does not entail $F : [l, u]$ iff $EQ_1(P, F)$ is solvable or $EQ_2(P, F)$ is solvable.* Therefore we get the following complexity bounds for the entailment problem.

Proposition 3 *The entailment problem for dp-programs is co-NP-complete.*

We note, in fact, that in Theorem 1 and Proposition 3 lower bounds hold for the class of simple p-programs.

3 Interval fixpoint and why it is not enough

The main goal of our study is to provide a precise mathematical description of the set $Mod(P)$ of the models, given a dp-program P. In this section, we discuss the fixpoint procedure for non-disjunctive p-programs, and establish its shortcomings with respect to the model theory outlined above.

Prior work on p-programs used a straightforward fixpoint procedure, which we reproduce below to find the probability intervals for each atom/basic formula of a p-program P. We noticed that in some cases the fixpoint procedure does not result in the exact description of $Mod(P)$ when the model-theoretic semantics of Section 2.2 is considered.

To simplify discussion and provide better intuition, this section mainly discusses p-programs (i.e., non-disjunctive programs), and, in fact, simple p-programs and follows [7,6] in the presentation. In the end of the section we generalize the definition of fixpoint to dp-programs. In all the cases, the structure of $Mod(P)$ turns out to be more complex than what the fixpoint procedure can compute.

The fixpoint semantics is defined on *atomic functions* and *formula functions.*

Definition 11 *Let* $\mathcal{C}[0,1]$ *denote the set of all subintervals of the interval* $[0,1]$. *An* atomic function *is a mapping* $f : B_L \rightarrow \mathcal{C}[0,1]$.

A formula function h *is a mapping* $h : bf(B_L) \rightarrow \mathcal{C}[0,1]$.

Given a set $\mathcal{F} \subseteq bf(B_L)$ *a* restricted formula function *is a mapping* $f_{\mathcal{F}} : \mathcal{F} \rightarrow \mathcal{C}[0,1]$.

Intuitively atomic and formula functions assign probability intervals to atoms and basic formulas: $h(F) = [l,u]$ can be interpreted as the statement `"probability of formula` F `lies in the interval` $[l,u]$`"`.

Definition 12 *Each formula function* $h_{\mathcal{F}}$ *induces a set* $\mathcal{LL}(h_{\mathcal{F}})$ *of linear inequalities on the probabilities* $p_1, \ldots, p_M$ *of possible worlds.* $\mathcal{LL}(h_{\mathcal{F}})$ *consists of the following inequalities:*

- $l_F \leq \sum_{W_j \models F} p_j \leq u_F$, *for all* $F \in \mathcal{F}$, $h_{\mathcal{F}}(F) = [l_F, u_F]$;
- $\sum_{j=1}^{M} p_j = 1$;
- $p_j \geq 0$, *for all* $1 \leq j \leq M$.

Note that $\sum_{W_j \models F} p_j$ is the probability of F. Therefore, the first group of inequalities specifies, in terms of probabilities p_j of possible worlds, the fact that the probability of F must be between l_F and u_F. The equality $\sum_{j=1}^{M} p_j = 1$ simply states that the sum of probabilities of all possible worlds adds up to 1, while inequalities of the form $p_j \geq 0$ specify that probabilities of possible worlds are nonnegative.

Given a p-program P, two operators, S_P and T_P are defined. They map formula functions to formula functions in the following manner.

$P:$
$C_1 : (a \wedge b) : [0.5, 1] \longleftarrow .$
$C_2 : (a \wedge b) : [0, 0.5] \longleftarrow .$
$C_3 : (a \wedge c) : [0.5, 0.5] \longleftarrow .$
$C_4 : (b \wedge c) : [0.5, 0.5] \longleftarrow .$
$C_5 : (a \wedge b \wedge c) : [0.1, 0.2] \longleftarrow .$

$p_1 + p_2 = 0.5$
$p_1 + p_3 = 0.5$
$p_1 + p_4 = 0.5$
$0.1 \leq p_1 \leq 0.2$
$\sum_{i=1}^{8} p_i = 1$
$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8 \geq 0$

Fig. 1 Sample p-program P, and the set of inequalities $\mathcal{LL}(S_P)$ it induces.

Definition 13 *Given a basic formula F,*

$$S_P(h)(F) = \cap M_F,$$

where $M_F = \{\mu | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n \in P,$ and $(\forall 1 \leq i \leq n)(h(F_i) \subseteq \mu_i)\}$. If $M_F = \emptyset$ then $S_P(h)(F) = [0, 1]$.

The T_P operator is defined as follows:

$$T_P(h)(F) = [l_F, u_F],$$

where $l_F = \min\left(\sum_{W_j \models F} p_j\right)$, subject to $\mathcal{LL}(S_P(h))$ and
$u_F = \max\left(\sum_{W_j \models F} p_j\right)$, subject to $\mathcal{LL}(S_P(h))$.

Intuitively, S_P computes the intervals of formulas based on the p-clauses that fired. If the heads of all p-clauses are atoms, i.e., for simple p-programs, this computation is enough. However, for basic formulas some tightening is required.

Example 4 Consider a p-program P which has p-clauses
$(a \wedge b) : [0.6, 0.9] \leftarrow .$
$(a \wedge c) : [0.8, 0.9] \leftarrow .$
The S_P operator will produce $S_P(a \wedge b) = [0.6, 0.9]$, $S_P(a \wedge c) = [0.8, 0.9]$, but $S_P(a) = S_P(b) = S_P(c) = [0, 1]$, because no p-clauses in P have heads a, b or c.

At the same time, it is clear that $[0, 1]$ is not a tight interval for probability of a, b, or c. In, particular if $Prob(a) = 0$, then $Prob(a \wedge b) = 0 \notin [0.6, 0.9]$. Such observations, however, are not captured by the S_P operator.

To propagate the information obtained from the clauses that fired to all other clauses, the T_P operator is employed. The work of these operators is illustrated on the following example.

Example 5 Consider the p-program P shown in Figure 1. Let $h(F) = [0, 1]$ for all $F \in bf(B_L)$. $S_P(h)(a \wedge b) = [0, 0.5] \cap [0.5, 1] = [0.5, 0.5]$. $S_P(h)(a \wedge c) = [0.5, 0.5]$; $S_P(h)(b \wedge c) = [0.5, 0.5]$ and $S_P(h)(a \wedge b \wedge c) = [0.1, 0.2]$. To compute $T_P(h)$ we first construct $\mathcal{LL}(S_P(h)$. Let $W_1 = \{a, b, c\}$, $W_2 = \{a, b\}$, $W_3 = \{a, c\}$ and $W_4 = \{b, c\}$. The set of inequalities $\mathcal{LL}(S_P)(h)$ is shown in Figure 1 (for simplicity we replace constraints of the form $a \leq X \leq a$ with $X = a$).

$a : [0.2, 0.4] \longleftarrow .$	(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. 2 Sample P-programs.

Combining the first three constraints with the fifth we get $2p_1 - 0.5 = p_5 + p_6 + p_7 + p_8$ or $p_1 = 0.25 + p_5 + p_6 + p_7 + p_8$. Because all $p_i \geq 0$, $\min(p_1)$ subject to the latter constraint is 0.25 (when all p_5,p_6,p_7,$p_8 = 0$). However, this contradicts the fourth constraint above which says, in particular $p_1 \leq 0.2$. Thus, $\mathcal{LL}(h)(S_P)$ has no solutions.

Example 6 Consider the p-program $P' = P - \{C_5\}$. The computation of $S_{P'}(h)$ will be the same as in the previous example, except $S_{P'}(h)(a \wedge b \wedge c) = [0, 1]$. Now, $T_{P'}(h)(a \wedge b \wedge c)$ is defined: $\min(p_1)$ subject to $\mathcal{LL}(S_{P'})(h)$ is 0.25 (see previous example for derivation). $\max(p_1) = 0.5$ and it is reached when $p_2 = p_3 = p_4 = 0$. Thus, $T_{P'}(h)(a \wedge b \wedge c) = [0.25, 0.5]$.

From [21] we know that the set of all formula functions over $bf(B_L)$ forms a complete lattice $\mathcal{FF}$ w.r.t. the subset inclusion: $h_1 \leq h_2$ iff $(\forall F \in bf(B_L))(h_1(F) \supseteq h_2(F))$. The bottom element $\perp$ of this lattice is the function that assigns $[0, 1]$ interval to all formulas, and the top element $\top$ is the atomic function that assigns $\emptyset$ to all formulas. Ng and Subrahmanian show that T_P is monotonic [21] w.r.t. $\mathcal{FF}$.

The iterations of T_P are defined in a standard way:

1. $T_P^0 = \perp$;
2. $T_P^{\alpha+1} = T_P(T_P^{\alpha})$, where $\alpha + 1$ is the successor ordinal whose immediate predecessor is α;
3. $T_P^{\lambda} = \sqcup\{T_P^{\alpha} | \alpha \leq \lambda\}$, where λ is a limit ordinal.

To connect model theory and fixpoint, we need to give the definition of satisfaction of a formula function by a p-interpretation.

Definition 14 *Let I be a p-interpretation over some Herbrand base B_L and h be a formula function over $bf(B_L)$. Then, $I \models h$ iff for all $F \in bf(B_L)$, $I(F) \in h(F)$ and there exists WP, s.t., WP satisfies $\mathcal{LL}(h)$ and $I = I_{WP}$.*

Ng and Subrahmanian show that, the least fixpoint $lfp(T_P)$ of the T_P operator is reachable after a finite number of iterations ([21], Theorem 2). They also show that if a p-program P is consistent, then $\mathcal{I}(lfp(T_P))$, the set of all p-interpretations satisfying $lfp(T_P)$ contains $Mod(P)$ ([21,22] Corollary 3). For simple p-programs $T_P = S_P$ and therefore can be computed in polynomial time.

At the same time, the inverse of the last statement, is not true, as evidenced by the following examples. First, consider a simple p-program P_1 shown in Figure 2.

Proposition 4 *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 turns out, that it is possible for a p-program to have a well-defined fixpoint but be inconsistent. Indeed, consider p-program P_2 from Figure 2.

Proposition 5 *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 4. 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 in this case $lfp(T_P)$ specifies the semantics of a simple p-program as the set of ap-interpretations inside a single N-dimensional parallelotope[5] (or, *N-parallelotope, for short*) 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-parallelotope, as evidenced by the following proposition.

Proposition 6 *If the atoms in B_L for P_1 (Figure 2) are ordered as a, b, then $Mod(P_1) = [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$.*

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

Consider an ap-interpretation I such that $I \models P_1$. 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 $I \in [0.2, 0.3) \times [0.2, 0.3]$.

[5] The extension of the 3D parallelepiped to N dimensions.

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.■

Looking at the proofs of the propositions above we see that the reason for the "bad" behavior of $lfp(T_P)$ lies in the computation of the S_P operator, namely, in the determination when p-clauses fire. By definition of S_P, a p-clause C fires if current valuation for each basic formula in the body of the clause is a subinterval of its annotation in the clause. Consider, for example a clause $C : F : \mu \longleftarrow G : \mu'$ and some formula function (valuation) h, such that $h(G) \not\subseteq \mu'$ but $h(G) \cap \mu' \neq \emptyset$. This clause will not fire. However, any p-interpretation $I \models C$ such that $I(G) \in h(G) \cap \mu'$, satisfies the body of the clause, and thus, must satisfy its head, i.e., we must have $I(F) \in \mu$. This extra restriction on the probability range of F is not captured by the S_P computation.

The definitions of operators S_P and T_P and fixpoint $lfp(T_P)$ can be generalized naturally to dp-programs. Only operator S_P should be changed slightly. Let P be a dp-program and h be a formula function.

Definition 15 *Given a basic formula G,*

$$S_P(h)(G) = \cap M_G,$$

where $M_G = \{\nu_j \mid G_1 : \nu_1 \vee \ldots \vee G_j : \nu_j \vee \ldots \vee G_k : \nu_k \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n \in P, G_j = G, (\forall 1 \leq r \leq k)($ *if* $r \neq j$ *then* $h(G_r) \cap \nu_r = \emptyset)$ *and* $(\forall 1 \leq i \leq n)(h(F_i) \subseteq \mu_i)\}$.

If $M_G = \emptyset$ *then* $S_P(h)(G) = [0, 1]$.

Informally, a member $G : \nu$ of a disjunction in a head of a clause takes part in the computation of probability bounds for G when all other members of the disjunction are inconsistent with h. The definitions of the operator T_P and fixpoint $lfp(T_P)$ do not change.

The results of Ng and Subrahmanian ([21,22]) on fixpoint remain valid for dp-programs. In particular, the least fixpoint $lfp(T_P)$ of the T_P operator is reachable after a finite number of iterations and $Mod(P) \subseteq \mathcal{I}(lfp(T_P))$. For simple dp-programs fixpoint $lfp(T_P)$ can be computed in polynomial time.

Example 7 Consider a simple dp-program P which has clauses
$C_1 = a : [0.3, 0.5] \vee b : [0.6, 0.8] \leftarrow .$

$C_2 = a : [0.5, 0.7] \vee b : [0.8, 0.9] \leftarrow c : [0.3, 0.5]$.
$C_3 = a : [0.1, 0.4] \leftarrow .$
$C_4 = c : [0.2, 0.4] \leftarrow .$

Then the first application of T_P ($= S_P$) to the bottom $\perp$ gives $T_P(a) = [0.1, 0.4], T_P(b) = [0, 1]$ and $T_P(c) = [0.2, 0.4]$. Since $[0.5, 0.7] \cap [0.1, 0.4] = \emptyset$ and $[0.2, 0.4] \subseteq [0.3, 0.5]$ the clause C_2 is fired on the next step and we get $T_P^2(a) = [0.1, 0.4], T_P^2(b) = [0.8, 0.9]$ and $T_P^2(c) = [0.2, 0.4]$. Clause C_1 is never fired, so fixpoint $lfp(T_P) = T_P^2(\perp) = [0.1, 0.4] \times [0.8, 0.9] \times [0.2, 0.4]$.

Let I^a be a model of P. Since $I^a \models C_1$ then $I^a(a) \in [0.3, 0.5] \cap [0.1, 0.4] = [0.3, 0.4]$ or $I^a(b) \in [0.6, 0.8] \cap [0.8, 0.9] = [0.8, 0.8]$. Therefore, $Mod(P) = [0.3, 0.4] \times (0.8, 0.9] \times [0.2, 0.4] \cup [0.1, 0.4] \times [0.8, 0.8] \times [0.2, 0.4] \neq lfp(T_P)$.

Based on the negative examples presented above we observe the following. The fixpoint procedure described in this section always returns a single interval as the set of possible probabilities for a basic formula. At the same time, model-theoretic semantics of dp-programs (and even p-programs) demands that in some cases this set be a more complex object — a union of closed, open and/or semi-open subintervals of [0,1]. In [6] and [7] we have described $Mod(P)$ for simple p-programs and p-programs respectively. In this paper, we extend the class of programs for which similar mathematical descriptions of $Mod(P)$ are possible to dp-programs.

4 Possible Worlds Semantics

We ask ourselves: given a p-program P, how do we give an exact description of $Mod(P)$? In this section we first answer this question for the general case of dp-programs. Then we show how in the case of simple p-programs we can obtain a simplified description of $Mod(P)$.

4.1 Semantics of dp-programs

Let us revisit the observations we made in Section 3 about the reasons why fixpoint computation has failed to produce the exact description of $Mod(P)$. Recall that $lfp(T_P)$ for a p-program P is a *formula function*, which specifies a single closed probability interval for each basic formula. Geometrically, it corresponds to an M-parallelotope ($M = |bf(B_L)|$). As seen from the examples in Section 3, the bodies of p-programs can be used to "pluck" points, or even intervals from $Mod(P)$ for a specific basic formula. This means that $Mod(P)$ is no longer a solid M-parallelotope, rather, it can have various portions missing. Thus, formula functions are inadequate as exact descriptors of $Mod(P)$.

Let us now construct $Mod(P)$ for a dp-program P. Remember, that the key reason why fixpoint procedure deviated from model theory in our examples in Section 3 was the fact that certain p-clauses could not be fired, whereas, there were p-interpretations which satisfied the bodies. The fixpoint procedure established the satisfaction of each p-clause in the p-program, but it did not track the *the reasons* why each p-clause was satisfied.

Recall that we have defined two sets of models for a dp-program P: $Mod(P)$ – models in the p-interpretation space and $WMod(P)$ – models in the probability density function space. In this section, we concentrate on building $WMod(P)$ given a dp-program P. Our first step is to capture the reasons for the satisfaction of clauses in a dp-program. To that end, given a dp-program P we define a set of systems of inequalities called $INEQ(P)$ as follows.

Definition 16 *Let P be a dp-program over the Herbrand base $B_L = \{A_1, \ldots, A_N\}$, and let $\mathcal{W} = (W_1, \ldots, W_M)$, $M = 2^N$ be an enumeration of all subsets of B_L. With each W_j, $1 \leq j \leq M$ we associate a variable p_j with domain $[0, 1]$. Let C be a disjunctive p-clause in P of the form*

$G_1 : [L_1, U_1] \vee \ldots \vee G_k : [L_k, U_k] \longleftarrow F_1 : [l_1, u_1] \wedge \ldots \wedge F_n : [l_n, u_n].$

where $k \geq 0$, $n \geq 0$ and $k + n > 0$.

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

$INEQ(C) = T(C) \cup F(C)$, *where*

$T(C) = \Big\{\{L_i \leq \sum_{W_j \models G_i} p_j \leq U_i\} | 1 \leq i \leq k\Big\}$

If k=0 then $T(C) = \emptyset$.

$F(C) = \Big\{\Big\{\sum_{W_j \models F_i} p_j < l_i\Big\} | 1 \leq i \leq n\Big\} \cup \Big\{\Big\{\sum_{W_j \models F_i} p_j > u_i\Big\} | 1 \leq i \leq n\Big\}.$

If n=0 then $F(C) = \emptyset$.

Let $P = \{C_1, \ldots, C_s\}$ and $\alpha_0 = \{\sum_{j=1}^{M} p_j = 1\}$. Then, $INEQ(P)$ is defined as follows:

$INEQ(P) = \{\alpha_0 \cup \alpha_1 \cup \ldots \cup \alpha_s \mid \alpha_i \in INEQ(C_i), 1 \leq i \leq s\}.$

For a system of inequalities $\alpha \in INEQ(P)$ let $Sol(\alpha)$ denote the set of all solutions of α.

Informally, $INEQ(P)$ is constructed as follows: for each p-clause C in the program we select the reason why it is true. The reason/evidence is either the statement that the head of the clause is satisfied, or that one of the conjuncts in the body is not. The former is captured in the systems from $T(C)$ which specify that one of the disjuncts in the head must be true, while the latter is captured in the systems from $F(C)$, which specify that one of the conjuncts is false. The set $INEQ(P)$ represents all possible systems of such evidence/restrictions on probabilities of basic formulas. Solutions of any system of inequalities in $INEQ(P)$ 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 3 *Consider a p-clause $C = (G_1 : [L_1, U_1] \vee \ldots \vee G_k : [L_k, U_k] \longleftarrow F_1 : [l_1, u_1] \wedge \ldots \wedge F_n : [l_n, u_n])$.*

A world probability function $WP \in WMod(C)$ iff there is a system of inequalities $\alpha \in INEQ(C)$ such that $\mathcal{P}_C = \{p_j = WP(W_j) | j = 1, \ldots, M\} \in Sol(\alpha)$.

Proof. $\Rightarrow$ Let $WP \in WMod(C)$ and $\mathcal{P}_C = \{p_j = WP(W_j), 1 \leq j \leq M\}$. Then $I_{WP} \in Mod(C)$, i.e. $I_{WP} \models C$. By Definition 7 at least one of two following assertions holds: (i) $I_{WP} \models G_1 : [L_1, U_1] \vee \ldots \vee G_k : [L_k, U_k]$ or (ii) $I_{WP} \not\models F_1 : [l_1, u_1] \wedge \ldots \wedge F_n : [l_n, u_n]$.

In case (i) $I_{WP} \models G_i : \nu_i$ for some $i \in [1, k]$ and $L_i \leq I_{WP}(G_i) \leq U_i$. Since $I_{WP}(G_i) = \sum_{W_j \models G_i} WP(W_j)$, then the system of two inequalities $\alpha = \{L_i \leq \sum_{W_j \models G_i} p_j \leq U_i\} \in T(C)$ hold and $\mathcal{P}_C \in Sol(\alpha)$.

In case (ii) $I_{WP} \not\models F_i : [l_i, u_i]$ for some $i \in [1, n]$. Hence $I_{WP}(F_i) < l_i$ or $I_{WP}(F_i) > u_i$. Then $\mathcal{P}_C \in Sol(\alpha)$ for $\alpha = \{\sum_{W_j \models F_i} p_j < l_i\} \in F(C)$ or for $\alpha = \{\sum_{W_j \models F_i} p_j > u_i\} \in F(C)$, respectively.

So, in all the cases there is a system of inequalities $\alpha \in INEQ(C)$ such that $\mathcal{P}_C \in Sol(\alpha \cup \{\sum_{j=1}^{M} p_j = 1\})$.

$\Leftarrow$ Let WP be a world probability function, $\mathcal{P} = \{p_j = WP(W_j), 1 \leq j \leq M\}$ and let $\mathcal{P}$ be a solution of a system inequlities $\alpha \in INEQ(C \cup \{\sum_{j=1}^{M} p_j = 1\})$. Since $INEQ(C) = T(C) \cup F(C)$ then (i) $\mathcal{P}$ is a solution of some $\alpha \in T(C)$ or (ii) $\mathcal{P}$ be a solution of some $\alpha \in F(C)$.

In case (i) $\alpha = \{L_i \leq \sum_{W_j \models G_i} p_j \leq U_i\}$ for some $i \in [1, k]$. Since $I_{WP}(G_i) = \sum_{W_j \models G_i} p_j$, the two inequalities $L_i \leq I_{WP}(G_i) \leq U_i\}$ hold. Then $I_{WP} \models G_i : [L_i, U_i]$ and by the definition 7 $I_{WP} \models C$.

In case (ii) α has a form of $\{\sum_{W_j \models F_i} p_j < l_i\}$ or $\{\sum_{W_j \models F_i} p_j > u_i\}$ for some $i \in [1, n]$. The sum in the left sides of the both inequalities is equal to $I_{WP}(F_i)$. Hence we get $I_{WP}(F_i) < l_i$ or $I_{WP}(F_i) > u_i$. In the both cases $I_{WP} \not\models F_i : [l_i, u_i]$. Then by the definition 7 $I_{WP} \not\models F_1 : [l_1, u_1] \wedge \ldots \wedge F_i : [l_i, u_i] \wedge \ldots F_n : [l_n, u_n]$ and $I_{WP} \models C$.

Therefore, in all the cases $WP \in WMod(C)$.■

Theorem 2 *A world probability function WP is a w-model of a dp-program P **iff** there exists a system of inequalities $\alpha \in INEQ(P)$ such that $\mathcal{P} = \{p_j = WP(W_j) | j = 1, \ldots, M\} \in Sol(\alpha)$.*

Proof follows from Lemma 3. Indeed, let $WP \models_w P$. Then for each clause $C \in P$, $WP \models_w C$. Then by Lemma 3, there exists $\alpha_C \in INEQ(C)$ such that $\mathcal{P}$ is a solution of α_C. Consider the following system of inequalities $\alpha = \cup_{C \in P} \alpha_C$. Then, $\alpha \in INEQ(P)$ by definition of the latter and $\mathcal{P}$ is its solution.

Conversely, let $\mathcal{P}$ be a solution for some $\alpha \in INEQ(P)$. By definition of INEQ(P), there exist systems of inequalities α_C for each $C \in P$ such that $\alpha = \cup_{C \in P} \alpha_C$. Then for each α_C, $\mathcal{P} \in Sol(\alpha_C)$. But then by Lemma 3 $WP \in WMod(C)$ for each clause C of P. Therefore, $WP \in WMod(P)$. ■

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

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

Let $r(P)$ be a number of clauses of P and $k(P)$ and $n(P)$ be the maximum number of basic formulas in a head and in a body of a rule in P, respectively. Then for every clause $C \in P$ the number of systems of inequalities in $INEQ(C)$ is bounded by $(k(P) + 2n(P))$. Therefore, $|INEQ(P)| \leq (k(P) + 2n(P))^{r(P)}$.

The solution of each system $\alpha \in INEQ(P)$ is a convex $M-1$-dimensional[6] (in the general case) polyhedron. Given a solution WP of some $\alpha \in INEQ(P)$, I_{WP} is obtained via a linear transformation. Because linear transformations preserve convexity of regions, we can make the following statement about the geometry of the set $Mod(P)$.

Corollary 2 *Given a dp-program P over the Herbrand base $B_L = \{A_1, \ldots, A_N\}$, $Mod(P)$ is a union of $S \leq (k(P)+2n(P))^{r(P)}$, not necessarily disjoint, convex polyhedra.*

This corollary provides an exponential, in the size of the p-program, upper bound on the number of *possibly disjoint* components of $Mod(P)$. Below we show that even for simple p-programs this exponential bound cannot be substantially decreased.

4.2 Semantics of Simple Disjunctive P-programs: Characterization of Models

In the general case $INEQ(P)$ is a complex construct, and using it to explicitly build $WMod(P)$ is difficult. However, when we consider the class of simple dp-programs, it turns out that it is possible to use it to define $Mod(P)$ directly, rather than the $WMod(P)$, which simplifies the description of the set of models of a program significantly.

Let P be a simple dp-program over the Herbrand base $B_L = \{A_1, \ldots, A_N\}$. Informally, P defines some restrictions on possible probabilities of the atoms of B_L. So, it is natural to use in the definition the semantics of P probabilities of atoms only. Recall that we defined an *ap-interpretation* I^a as a function $I^a : B_L \rightarrow [0, 1]$ and $Mod(P) = \{I^a | I^a \models P\}$. Using this, we can simplify the definition of $INEQ(P)$ for simple dp-programs.

Definition 17 *Let P be a simple dp-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 $C \equiv D_1 : [L_1, U_1] \vee \ldots \vee D_k : [L_k, U_k] \longleftarrow B_1 : [l_1, u_1] \wedge \ldots \wedge B_n : [l_n, u_n]$, $k \geq 0, n \geq 0, k + n > 0$, be a clause of P.*

The family of systems of inequalities induced by C, *denoted $INEQ(C)$ is defined as follows:*
$INEQ(C) = T(C) \cup F(C)$;
$T(C) = \{\{L_j \leq x_{D_j} \leq U_j \mid 1 \leq j \leq k\}\}$ *(if k=0 then $T(C) = \emptyset$);*

[6] Because $p_1 + \ldots + p_M = 1$ is present in every $\alpha \in INEQ(P)$, the dimensionality of $Sol(\alpha)$ cannot be more than $M - 1$.

$F(C) = \{\{x_{B_i} < l_i\} | 1 \leq i \leq n\} \cup \{\{x_{B_i} > u_i\} | 1 \leq i \leq n\}$ *(if n=0 then* $F(C) = \emptyset$*).*
Let $P = \{C_1, \ldots, C_s\}$*. The family* $INEQ(P)$ *of* systems of inequalities *is defined as* $INEQ(P) = \{\alpha_1 \cup \ldots \cup \alpha_s | \alpha_i \in INEQ(C_i), 1 \leq i \leq s\}$.

The key difference between Definition 16 of $INEQ(P)$ in general case and Definition 17 is that for simple dp-programs we can directly use variables encoding probabilities of atoms in B_L rather than probabilities of possible worlds in a world probability density function. Thus, $INEQ(P)$ for simple dp-programs is built in terms of (a)p-interpretations rather than in terms of underlying world probability density functions. Another important observation is that now, all inequalities in all systems from the definition above involve only one variable.

As above, 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 we observe that

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

As with the general case, the set $INEQ(P)$ represents all possible systems of restrictions on probabilities of atoms of B_L whose solutions satisfy every clause of P. Not all individual systems of inequalities need to have solutions, but similarly to the general case, we show that $INEQ(P)$ *captures all the systems that do.*

Lemma 4 *Let* $C = D_1 : [L_1, U_1] \vee \ldots \vee D_k : [L_k, U_k] \longleftarrow B_1 : [l_1, u_1] \wedge \ldots \wedge B_n : [l_n, u_n]$*, be a disjunctive p-clause of* P *and* I^a *be a* $ap-interpretation$ *(both over the same Herbrand Base* B_L*). Then* $I^a \models C$ *iff* $\{x_A = I^a(A)\} \in Sol(\alpha)$ *for some* $\alpha \in INEQ(C)$*.*

Proof. $\Rightarrow$ Let $I \models C$. Two cases need to be considered.
1. $I^a \models D_1 : [L_1, U_1] \vee \ldots \vee D_k : [L_k, U_k]$. Then, by definition of satisfaction, $I^a \models D_j : [L_j, U_j]$ for some $j \in [1, k]$. This means that $[L_j \leq I^a(D_j) \leq U_j$. Notice that the following systems of inequalities: $\alpha = \{L_j \leq x_{D_j} \leq U_j\}$ is contained in $T(C)$ and thus $\alpha \in INEQ(C)$. But then $\{x_A = I^a(A) | A \in B_L\}$ is a solution of α since $x_{D_j} = I^a(D_j)$.
2. $I^a \not\models B_1 : [l_1, u_1] \wedge \ldots \wedge B_k : [l_k, u_k]$. Then, there exists such $1 \leq j \leq k$ that $I^a(B_j) \notin [l_j, u_j]$, i.e., either $I^a(B_j) < l_j$ or $I^a(B_j) > u_j$. In the first case, $\{x_A = I^a(A), x_i = I^a(B_i) | 1 \leq i \leq k\}$ is a solution of the system $\alpha' = \{x_{B_j} \leq l_j\} \in F(C)$, and in the second case, it will be a solution of the system $\alpha'' = \{x_{B_j} \geq u_j\} \in F(C)$ (note that both systems consist of a single inequality, limiting the value of x_{B_j} *only*). As $F(C) \subseteq INEQ(C)$, the statement of the lemma is true.
$\Leftarrow$ Let $\alpha \in INEQ(C)$ and let $U = \{x_{A_i} = p_{A_i} | A_i \in B_L\}$ be a solution of α. Consider a ap-interpretation I^a, s.t., $I^a(A_i) = p_{A_i}$. We show that $I^a \models C$.

Since $INEQ(C) = T(C) \cup F(C)$ then either $\alpha \in T(C)$ or $\alpha \in F(C)$.

Let $\alpha \in T(C)$. Then $\alpha = \{L_j \leq x_{D_j} \leq U_j\}$ for some $j \in [1, k]$. As U is a solution of α, $L_j \leq p_A \leq U_j$. Therefore, $I^a(D_j) \in [L_j, U_j]$ and $I^a \models C$.

Consider now the last remaining case, namely, $\alpha \in F(C)$. Then α has a form of $\{x_{B_j} < l_j\}$ or $\{x_{B_j} > u_j\}$. Without loss of generality, consider the former case. As U is the solution of α, $x_{B_j} < l_j$ and therefore $I^a(B_j) \notin [l_j, u_j]$. But then, $I^a \not\models B_1 : [l_1, u_1] \wedge \ldots \wedge B_k : [l_k, u_k]$, which means that $I^a \models C$. ■

Theorem 3 *A ap-interpretation I^a is a model of a simple dp-program P **iff** there exists a system of inequalities $\alpha \in INEQ(P)$ such that $X = \{x_A = I^a(A)\} \in Sol(\alpha)$.*

Proof follows from Lemma 4. Indeed, let $I^a \models P$. Then for each clause $C \in P$, $I^a \models C$. Then by Lemma 4, there exists $\alpha_C \in INEQ(C)$ such that X is a solution of α_C. Consider the following system of inequalities $\alpha = \cup_{C \in P} \alpha_C$. Then, $\alpha \in INEQ(P)$ by definition of the latter and X is its solution.

Conversely, let $X = \{x_A = b_A | A \in B_L\}$ be a solution for some $\alpha \in INEQ(P)$. By definition of INEQ(P), there exist systems of inequalities α_C for each $C \in P$ such that $\alpha = \cup_{C \in P} \alpha_C$. Then for each α_C, $X \in Sol(\alpha_C)$. But then ap-interpretation I^a defined as $I^a(A) = x_A$ for all $A \in B_L$ satisfies every clause C of P. Therefore, I^a is a model of P. ■

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

Corollary 3 $Mod(P) = \bigcup_{\alpha \in INEQ(P)} Sol(\alpha)$

Let $r(P)$ be the number of clauses of P and $k(P)$ and $n(P)$ be the maximum number of basic formulas in a head and in a body of a rule in P, respectively. Then for every clause $C \in P$ the number of systems of inequalities in $INEQ(C)$ is bounded by $(k(P) + 2n(P))$. Therefore, $|INEQ(P)| \leq (k(P) + 2n(P))^{r(P)}$.

The solution of each system $\alpha \in INEQ(P)$ is a convex N-dimensional (in the general case) parallelotope.

Corollary 4 *For a simple dp-program P, $Mod(P)$ is a union of at most $M(P)$ (not necessarily disjoint) N-parallelotope, where $M(P) = (k(P) + 2n(P))^{r(P)}$.*

This Corollary provides an exponential, in the size of the p-program, upper bound on the number of parallelotopes in the set $Mod(P)$.

Consider as the simplest example the subclass of simple dp-programs consisting of facts only, i.e. of clauses with empty bodies. Let P be such a program consisting of clauses $C_1, \ldots, C_s$ and let $C_i = D_1^i : [L_1^i, U_1^i] \vee \ldots \vee D_{k_i}^i : [L_{k_i}^i, U_{k_i}^i] \longleftarrow$. Then $Mod(P)$ can be defined by the following procedure **GenModFacts**.

1) Fix the set of indices $J = \{(j_1, \ldots, j_s) \mid 1 \leq j_r \leq k_r \textit{ for all } r \in [1, s]\}$.

2) For each $\alpha = (j_1, \ldots, j_s) \in J$ and $A_i \in B_L$ choose the subset of clause indices $J_\alpha(A_i) = \{k \mid D_{j_k}^k = A_i\}$,

3) If $J_\alpha(A_i) \neq \emptyset$ then let $\nu_\alpha(A_i) = [L_i, U_i]$ where $L_i = \max\{L^k_{j_k} \mid k \in J_\alpha(A_i)\}$ and $U_i = \min\{U^k_{j_k} \mid k \in J_\alpha(A_i)\}$.
If $J_\alpha(A_i) = \emptyset$ then let $\nu_\alpha(A_i) = [0, 1]$.
4) If for all $i \in [1, N]$ the interval $\nu_\alpha(A_i)$ is nonempty, i.e. $L_i \leq U_i$, then let $Mod_\alpha(P) = \times_{i=1}^{N} \nu_\alpha(A_i)$, otherwise $Mod_\alpha(P) = \emptyset$.
5) Return $Mod(P) = \bigcup_{\alpha \in J} Mod_\alpha(P)$.

The procedure **GenModFacts** analyzes $|J| = \prod_{i=1}^{s} k_i$ possible choices of α and sets of models (parallelotopes) $Mod_\alpha(P)$. The relationships of equality, inclusion and nonempty intersection are possible among these parallelotopes.

The following example shows that $Mod(P)$ for simple dp-programs with empty bodies of clauses can include an exponential number of disjoint parallelotopes.

Example 8 Let P consist of $s = N$ clauses of the form $C_i = A_1 : [L^i_1, U^i_1] \vee \ldots \vee A_N : [L^i_N, U^i_N] \longleftarrow .$ and $[L^i_r, U^i_r] = [(i-1)/N, (2(i-1)+1)/2N]$ for all $i \in [1, N]$ and $r \in [1, N]$. Then for any two permutations α_1 and α_2 of $(1, 2, \ldots, N)$ the sets of models $Mod_{\alpha_1}(P)$ and $Mod_{\alpha_2}(P)$ do not intersect, while for each α which is not a permutation $Mod_\alpha = \emptyset$. So, $Mod(P)$ consists of $N!$ disjoint parallotope.

We can show that an exponential bound on the number of disjoint parallelotopes in $Mod(P)$ cannot be substantially decreased even for non-disjunctive simple p-programs.

Example 9 Consider p-program P' 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: $\alpha^1_i = \{0 \leq x_{b_i} < 0.2\}$, $\alpha^2_i = \{0.3 < x_{b_i} \leq 1\}$, and $\alpha^3_i = \{0.2 \leq x_{b_i} < 0.3; x_a = 0\}$. Since α^3_i is inconsistent with $INEQ(1)$, each consistent set of inequalities in $INEQ(P')$ can be represented as $\{x_a = 1\} \cup \bigcup_{i=1}^{n} \alpha^{j_i}_i$ 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 $INEQ(P')$ sets $Sol(\alpha)$ and $Sol(\alpha')$ are disjoint. So, $Mod(P')$ consists of 2^n disjoint n-parallelotopes. At the same time $f(P') = n + 1,\ r(P') = n,\ k(P') = 1$ and a bitwise representation of P_3 takes only $O(n \log n)$ bits.

4.3 Semantics of Simple dp-programs: Explicit Computation of Models

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

The construction from the 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 Example 9 illustrates the worst case cannot be avoided. However, it is 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 dp-program. This approach breaks the solution space into *disjoint* components and individually computes each such component.

Consider a simple dp-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 as follows $Prb_P(A) = \{\langle l, -\rangle | A : [l, u] \in AT(P)[A]\} \cup \{\langle u, +\rangle | A : [l, u] \in AT(P)[A]\} \cup \{\langle 0, -\rangle, \langle 1, +\rangle\}$. 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 $a < b$, and $\langle a, -\rangle < \langle a, +\rangle$. Consider now $Prb_P(A) = \{\beta_1 = \langle 0, -\rangle, \beta_2, \ldots, \beta_m = \langle 1, +\rangle\}$ where the sequence $\beta_1, \ldots, \beta_m$ is in ascending order. Using the set $Prb_P(A)$ we 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 Table 1.

Table 1 Determination of segments in $SEG(A)$

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

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 7 *Let P be a simple dp-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 $s_i \cap s_j = \emptyset$.
*(2) Consider some $1 \leq i \leq m-1$. Let $x, y \in s_i$ and let I_1 and I_2 be ap-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** $I_2 \models A : \mu$.*

Proof (1) The definition of segmentation $SEG_P(A)$ ensures that that any interval with an open end of the form $[a_i, a_{i+1})$ or (a_i, a_{i+1}) always is followed by some interval with a closed begin of the form $[a_{i+1}, a_{i+2}]$ or $[a_{i+1}, a_{i+2})$ (note, that the last interval in $SEG_P(A)$ *always* has a closed end, so intervals with an open must have a successor interval). Therefore all the points of $[0, 1]$ are included in $SEG_P(A)$. By the same reason any interval with a closed end of the form $[a_i, a_{i+1}]$ or $(a_i, a_{i+1}]$ always is followed by some interval with an open begin of the form $(a_{i+1}, a_{i+2}]$ or (a_{i+1}, a_{i+2}). Therefore, no point of $[0, 1]$ can be included into two segments of $SEG_P(A)$.
(2) Suppose that for some $A : [l, u] \in AT(P)[A]$ ap-interpretations I_1 and I_2 behave differently: $I_1 \models A : [l, u]$ and $I_2 \not\models A : [l, u]$. Then $l \leq x \leq u$ and $y < l$ or $y > u$. It follows by the definition of $SEG_P(A)$ that x is included in a segment s with lower bound l_s and upper bound u_s such that $l \leq l_s$ and $u_s \leq u$, while y is included in a segment s' with lower bound $l_{s'}$ and upper bound $u_{s'}$ that $l_{s'} < l$ or $u_{s'} < u$. Therefore $s \neq s'$ which contradicts the condition of (2). ■

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

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

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 an ap-interpretation. The following theorem shows that the set of all ap-interpretations satisfying P can be constructed from some "bricks" of $SEG(P)$.

Theorem 4
(1) Any two different parallelotopes of $SEG(P)$ do not intersect.
(2) For any parallelotope $J \in SEG(P)$ either $J \subseteq Mod(P)$, or $J \cap Mod(P) = \emptyset$.
(3) There exists a subset $S \subseteq SEG(P)$ such that $Mod(P) = \bigcup_{J \in S} J$.

Proof follows directly from proposition 7.

Example 10 Consider again program P_1 shown in Figure 2:
$a : [0.2, 0.4] \longleftarrow .$
$b : [0.2, 0.5] \longleftarrow .$
$b : [0.2, 0.3] \longleftarrow a : [0.2, 0.3].$
$b : [0.4, 0.5] \longleftarrow a : [0.3, 0.4].$
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 6 only 2 of them consist of models of P_3: $Mod(P_1) = [0.2, 0.3) \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$.

(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. 3 Algorithm GenMod for computing $Mod(P)$.

Theorem 4 suggests that $Mod(P)$ can be constructed using the algorithm *GenMod* described in Figure 3. 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 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 $(l^i + u^i)/2$ — this point is guaranteed to belong to the interval regardless of which ends of the interval are open or closed. 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 in the size of the set B_L of all atoms of P. Of course, it is possible that 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 the 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. For the same reason, in the worst case, exponential space is needed to store the output of *GenMod*. At the same time, the size of all $Prb_P(A)$ does not exceed the size of P itself (as, all values in $Prb_P(A)$ are explicitly found in P). Because of this, algorithm *GenMod* can work using only linear *working* or *internal memory*, requiring exponential memory only to write out the output.

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 dp-program P, let $lfp(T_P(A_i)) = sg_i$ and $NS(P) = \times_{i=1}^{N} sg_i$. From [21] (Lemma 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 $\{s | s \in S \text{ and } s \subseteq \mu\}$. We now define the notion of a reduct.

Definition 18 *Given a simple dp-program P, an atom $A \in B_L$ and an interval $\nu = \langle l, u \rangle \subseteq [0, 1]$, which can be open or closed on either end*[7]*,*

[7] I.e., $\langle \in \{[, (\}, \rangle \in \{],)\}$,.

we denote by $Reduct(P, A : \nu)$ a reduced program which results from P as follows:

(i) Delete from P any clause C whose head includes an atom $A : \mu$ such that $\nu \subseteq \mu$.
(ii) Delete from P any clause C whose body includes an atom $A : \mu$ such that $\mu \cap \nu = \emptyset$.
(iii) Delete from the body of any other rule each atom $A : \mu$ such that $\nu \subseteq \mu$.
(iv) Delete from the head of any other rule each atom $A : \mu$ such that $\mu \cap \nu = \emptyset$.

The following assertion relates the set of models of the original program P and the sets of models of its reducts.

Proposition 8
(1) Let $B_L = \{A_1, \ldots, A_N\}$ and $1 \leq j \leq N$. Then
$Mod(Reduct(P, A_j : \nu_j) \cup \{A_j : \nu_j \leftarrow .\}) = Mod(P \cup \{A_j : \nu_j \leftarrow .\}) = Mod(P) \cap [0,1]^{j-1} \times \nu_j \times [0,1]^{N-j}$.
(2) For each $s \in SEG_P(A)$ program $Reduct(P, A : \nu)$ does not include any atom of the form $A : \mu$.

Proof. (1) It is straightforward to check that each transformation (i)–(iv) preserves the set of models of $(P \cup \{A : \nu \leftarrow .\})$. (2) follows from the following fact: for each $s \in SEG_P(A)$ and $A : \mu \in AT(P)[A]$ either $s \subseteq \mu$ or $s \cap \mu = \emptyset$. ■

We extend now the reduction of P with respect to one atom $A : \nu$ to the reduction with respect to an atomic formula function $h : B_L \rightarrow C[0,1]$.

Definition 19 *Let P be a simple dp-program and let h be an atomic formula function. The reduct of program P w.r.t. h, denoted $Reduct(P, h)$ is*
$Reduct(P, h) = Reduct(Reduct(\ldots Reduct(Reduct(P, A_1 : h(A_1)), A_2 : h(A_2)) \ldots), A_N : h(A_N))$

Consider the algorithm ComputeReduct described below.

Algorithm ComputeReduct(P, h)

(1) **for each** $A \in B_L$ **do** $P := Reduct(P, A : h(A))$;
(2) **return** P.

The following assertion follows straightforward from Proposition 8.

Proposition 9
(1) The result of Algorithm ComputeReduct does not depend on the order of atoms of B_L in line (1).
(2) Algorithm ComputeReduct(P,h) computes $Reduct(P, h)$.
(3) Let $B_L = \{A_1, \ldots, A_N\}$ be some ordering of B_L. Then

$$Mod(Reduct(P, h) \cup \bigcup_{j=1}^{N} \{A_j : h(A_j) \leftarrow .\}) = Mod(P \cup \bigcup_{j=1}^{N} \{A_j : h(A_j) \leftarrow .\}) =$$

```
Algorithm GenModT(P:program, {A_1, ..., A_N}:atoms)
if P includes contradictory clause  ←. then return(∅)
else
  Sol := ∅;
  S := NS(P);      // compute fixpoint of Ng-Subrahmanian T_P operator
  if S = ∅ then return(∅)
  else // if NS(P) is not empty, proceed with computations
    P := Reduct(P, S);
    Seg := SEG(P, A_1) ∩ sg_1;    // the segmentation of A_1 inside
                          // the fixpoint of operator T_P
                          // (recall, sg_1 = lfp(T_P)(A_1))
// main loop
    for each s = ⟨a, b⟩ ∈ Seg do
      P' := Reduct(P, A_1 : s);
      if P' is empty then   Sol := Sol ∪ (s × (×_{i=2}^{N} [0, 1]))
      else // find the solution for the reduct
        RSol := GenModT(P', {A_2, ..., A_N});
        if RSol ≠ ∅ then   Sol := Sol ∪ (s × RSol)  end if
      end if   end do
end if   end if
return Sol;
```

Fig. 4 Algorithm GenModT for computing $Mod(P)$.

$$Mod(P) \cap (h(A_1) \times \ldots \times h(A_N)).$$

Figure 4 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 5 *Given a simple p-program P and an ordering $A_1, \ldots, A_N$ of B_L, algorithm GenModT returns the set $Mod(P)$.*

Proof The result follows in a straightforward way from Propositions 8 and 9.

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 $\mathsf{GenModT}$ 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 of$\mathsf{GenModT}$ is that it structures $Mod(P)$ in a form of a tree, corresponding to the way it recursively enumerates the solutions.

The advantages of $\mathsf{GenModT}$ over the naïve implementation of GenMod are demonstrated in the example of program P_1 (Fig. 2). In Example 10 we showed 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 $\mathsf{GenModT}$ $Seg = SEG_{P_1}(a) \cap (0.2, 0.4] = \{[0.2, 0.3), [0.3, 0.3], (0.3, 0.4]\}$ 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 6). Thus, $\mathsf{GenModT}$ tries only 3 bricks while GenMod will check all 25 bricks.

5 When Fixpoint is Enough

In this section we study subclasses of p-programs for which simpler procedures for determining $Mod(P)$ exist. In particular, we study when $Mod(P)$, as defined here, and $lfp(T_P)$, as defined for p-programs in [22] and for simple dp-programs in Section 3 coincide. We then address the problem of complexity of detecting that $Mod(P) = \mathcal{I}(lfp(T_P))$.

It turns out that it is possible to specify a necessary and sufficient condition for equivalence of $Mod(P)$ and $\mathcal{I}(lfp(T_P))$ for a general case of simple dp-programs.

To simplify the discussion, we assume that a simple dp-program P under consideration does not contain trivial clauses C which are satisfied by any ap-interpretation, i.e. with $Mod(C) = E^N$. In addition, we assume that for each clause $C \in P$ the following simple syntactic conditions hold:

1. each atom A occurs in the body of C at most once (multiple occurrences $A : \mu_1, \ldots, A : \mu_r$ can be replaced with one occurrence $A : \mu$ where $\mu = \mu_1 \cap \ldots \cap \mu_r$);

2. for each two occurrences $A : \nu_1$ and $A : \nu_2$ in the head of C their intervals do not intersect: $\nu_1 \cap \nu_2 = \emptyset$ (otherwise they can be replaced with $A : (\nu_1 \cup \nu_2)$);
3. if the head of C includes $A : \nu$ and the body of C includes $A : \mu$ then $\nu \subseteq \mu$ and $\mu - \nu \neq \emptyset$ (otherwise $A : \nu$ can be changed to $A : \nu \cap \mu$, if $\nu \cap \mu = \emptyset$ then this atom can be deleted from the head of C).

In the previous section the reduction $Reduct(P, A : \nu)$ of a simple dp-program P w.r.t. atom $A : \nu$ was defined (see Definition 18). We strengthen this reduct and define a new reduct, $Reduct1(P, A : \nu)$ by adding to the transformations (i)–(iv) of $Reduct(P, A : \nu)$ from Definition 18 two new transformations:

(v) Change each atom $A : \mu$ in the bodies of clauses of P to the atom $A : (\mu \cap \nu)$;
(vi) Delete from P any clause C such that the head of C contains an atom $A : \mu$, the body of C contains an atom $A : \mu_1$ and $\mu_1 \subseteq \mu$.

The reduction $Reduct1(P, h)$ of a simple dp-program P w.r.t. atomic formula function h is defined in the same way as $Reduct(P, h)$.

New transformations sometimes allow to decrease the size of the reduced program but do not change the set of its models. Because of this, the following proposition holds.

Proposition 10
(1) Let $B_L = \{A_1, \ldots, A_N\}$ and $1 \leq j \leq N$. Then
$Mod(Reduct1(P, A_j : \nu_j) \cup \{A_j : \nu_j \leftarrow .\}) = Mod(P \cup \{A_j : \nu_j \leftarrow .\}) = Mod(P) \cap [0,1]^{j-1} \times \nu_j \times [0,1]^{n-j}$.
(2) Let $B_L = \{A_1, \ldots, A_N\}$ be some ordering of B_L. Then
$Mod(Reduct1(P, h) \cup \bigcup_{j=1}^{N} \{A_j : h(A_j) \leftarrow .\}) = Mod(P \cup \bigcup_{j=1}^{N} \{A_j : h(A_j) \leftarrow .\}) = Mod(P) \cap (h(A_1) \times \ldots \times h(A_N))$.

We use $Reduct1(P, h)$ to define the subclass of strict simple dp-programs.

Definition 20 *A simple dp-program P is called* strict *if $Reduct1(P, lfp(T_P)) = \emptyset$.*

For p-programs strictness can be defined as follows.

Definition 21 *Let P be a p-program and let P' be the result of removing from P all p-clauses whose heads are satisfied by $lfp(T_P)$. A p-program P is called* strict *if the following condition holds: for each clause $C : F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots F_n : \mu_n$ in P', there exists an index $1 \leq i \leq n$, such that $lfp(T_P)(F_i) \cap \mu_i = \emptyset$.*

For the class of simple dp-programs, strictness can be efficiently checked and is a sufficient and necessary condition. This leads to polynomial-time upper bounds on entailment and consistency.

Theorem 6
(1) For a simple dp-program P deciding whether it is strict can be performed in polynomial time.
*(2) For a simple dp-program P, $Mod(P) = \mathcal{I}(lfp(T_P))$ **iff** P is strict.*

Proof of (1) follows from the fact that the fixpoint $lfp(T_P)$ and $Reduct1(P, lfp(T_P))$ are computable in polynomial time of the size of P.

(2) Suppose that $Reduct1(P, lfp(T_P)) = \emptyset$. Then from this equality and from proposition 10 (2) it follows that $Mod(Reduct1(P, lfp(T_P)) \cup \bigcup_{j=1}^{N}\{A_j : lfp(T_P)(A_j) \leftarrow .\}) = Mod(\bigcup_{j=1}^{N}\{A_j : lfp(T_P)(A_j) \leftarrow .\}) = (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N)) = Mod(P) \cap (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N))$. As $Mod(P) \subseteq (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N)) = \mathcal{I}(lfp(T_P))$, we conclude that $Mod(P) = (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N)) = \mathcal{I}(lfp(T_P))$.

Now suppose that $Reduct1(P, lfp(T_P)) \neq \emptyset$. Then $Reduct1(P, lfp(T_P))$ includes some clause C of the form $D_1 : \nu_1 \vee \ldots \vee D_k : \nu_k \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_n : \mu_n$. From the definition of $Reduct1(P, lfp(T_P))$ it follows that for each $1 \leq j \leq k$, the set $\alpha_j = lfp(T_p)(D_j) - \nu_j \neq \emptyset$ and for each $1 \leq i \leq n$, the set $\beta_i = \mu_i \cap lfp(T_p)(B_i) \neq \emptyset$. Besides, each atom $A \in B_L$ can occur in the head of C at most twice and in the body of C at most once.

Now define for every $A \in B_L$ an ap-interpretation $I^a(A)$ as follows. Let $lfp(T_P)(A) = [L, U]$.
Let $A : [l_1, u_1], A : [l_2, u_2], \ldots, A : [l_r, u_r]$ $(r \geq 0)$ be all occurrences of A in the head of C ordered from the left to right, i.e. $l_1 < u_1 < l_2 < u_2 < \ldots < l_r < u_r$.

(1) If $r \geq 2$ we choose two leftmost occurrences $A : [l_1, u_1]$ and $A : [l_2, u_2], l_1 \leq u_1 < l_2 \leq u_2$, and put $I^a(A) = (u_1 + l_2)/2$. From the conditions on the considered intervals it follows that $I^a(A) \in [L, U] = lfp(T_P)(A)$ and for each occurrence $A : [l_i, u_i]$ $(1 \leq i \leq r)$ in the head of C the value $I^a(A) \notin [l_i, u_i]$. Hence, $I^a(A) \not\models A : [l_1, u_1] \vee A : [l_2, u_2] \vee \ldots A : [l_r, u_r]$. On the other hand, if $A : [l, u]$ is the (only!) occurrence of A in the body of C then $l_1 \leq l \leq u_1 < l_2 \leq u$ and $I^a(A) \in [l, u]$. Therefore, $I^a(A) \models A : [l, u]$.
(2) If $r = 1$ and $A : [l_1, u_1]$ is the only occurrence of A in the head of C and $A : [l, u]$ is the occurrence of A in the body of C then the points (v) and (vi) of the definition of $Reduct1(P, A : [L, U])$ ensure that $([L, U] \cap [l, u]) \not\subseteq [l_1, u_1]$, so we can choose a point $I^a(A)$ in the interval $([L, U] \cap [l, u]) - [l_1, u_1]$.
(3) If $A : [l_1, u_1] = D_j : \nu_j$ is the only occurrence of A in the head of C and the body of C does not include A, then choose $I^a(A)$ to be the middle of the interval α_j.
(4) If $A : [l, u] = B_i : \mu_i$ is the only occurrence of A in the body of C and the head of C does not include A, then choose $I^a(A)$ to be the middle of the interval β_i.
(5) If C does not contain atom A then put $I^a(A) = (L + U)/2$.

It follows from the definition of I^a that for each $A \in B_L$ $I^a(A) \in lfp(T_P)(A)$. Besides, in all the cases above $I^a(B_i) \in \mu_i$, so $I^a \models B_i : \mu_i$ for all $1 \leq i \leq n$. On the other hand, for every $1 \leq j \leq k$ the value

$I^a(D_j) \notin \nu_j$, hence $I^a \not\models D_j : \nu_j$. Therefore, $I^a \not\models C$. On the other hand, $I^a \in \mathcal{I}(lfp(T_P))$. Hence, $\mathcal{I}(lfp(T_P)) \neq Mod(P)$. ■

From Theorem 6 we can obtain the following corollary.

Corollary 5 *Consistency and entailment problems are solvable in polynomial time for strict simple dp-programs.*

For the class of p-programs, strictness is a sufficient condition.

Theorem 7 *If a p-program P is strict, then $Mod(P) = \mathcal{I}(lfp(T_P))$.*

Proof. We know that $Mod(P) \subseteq \mathcal{I}(lfp(T_P))$. Suppose now, $I \in \mathcal{I}(lfp(T_P))$. We show that for every dp-clause $C \in P$ of the form $F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots F_n : \mu_n$, $I \models C$. If $C \in P - P'$, then $I(F) \in lfp(T_P)(F) \subseteq \mu$, and therefore, $I \models F : \mu$. If $C \in P'$, then, because C is strict, there exists an index i, that $lfp(T_P)(F_i) \cap \mu_i = \emptyset$. Then $I \not\models F_i : \mu_i$, and therefore $I \not\models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots F_n : \mu_n$ and $I \models C$.■

The following example shows that strictness is not a necessary condition for non-simple programs.

Example 11 Consider the following p-program P_7:
$a : [0.6, 0.8] \longleftarrow .$ $\quad$ $b : [0.6, 0.7] \longleftarrow .$ $\quad$ $d : [0.2, 0.3] \longleftarrow .$
$c : [0.4, 0.5] \longleftarrow (a \wedge b) : [0.65, 0.7] \wedge (b \vee d) : [0.5, 0.6]$.
$lfp(T_P)$ assigns intervals $[0.2, 0.7]$ and $[0.6, 1]$ to $a \wedge b$ and $b \vee d$ respectively, and therefore, P_7 is *not strict*. However, there exists no p-interpretation I which satisfies the first three rules and the body of the fourth rule: $I(b \vee d) \in [0.5, 0.6]$ implies, $I(b \vee d) = 0.6$ and $I(b) = 0.6$, while $I(a \wedge b) \in [0.65, 0.7]$ implies that $I(b) \geq 0.65$. Therefore, $Mod(P)$ coincides with $\mathcal{I}(lfp(T_P))$.

6 Related Work and Conclusions

Surverys. A survey of different approaches to probabilistic logic programming can be found in [8] and in [5]. Logic for reasoning with probabilistic data is described in [25] and [9]. A good introduction to imprecise probabilities is the book by Walley [29]. The possible worlds semantics for interval probabilities can be found in [3].

Annotated Logic Programs and PSAT-based logic programming. Kifer and Subrahmanian first considered generalized annotated logic programs (GAPs), i.e., logic programs with rules over $F : \mu$ clauses in [13]. In that work, μ were assumed to take their values from a semilattice of truth values. Theis language has been extended to explicitly treat μ values as interval probabilities by Ng and Subrahmanian[22,21,23]. As discussed in Section 1 the latter work is also a natural extension of Interval Probabilistic Satisfiability problem PSAT [11]: an instance of Interval PSAT is a p-program, in which all rules have no bodies.

The language we consider in this paper is a natural extension of the syntax of Ng and Subrahmanian (as well as Kifer and Subrahmanian) to allow for disjunctions in the heads of clauses. We show that for this, relatively simple language, the class of satisfying models (probabilistic interpretations) has a complex description: it is a union of a number of (closed, open, semi-open) intervals, obtained, solving an array of Interval PSAT problems. On the positive side, our results show how to compute the set of models of a p-program *precisely*. On the negative side, the complexity of the description and the computational complexity of the problem itself suggest that intervals may be inadequate as the means for specifying imprecision in probabilistic assessments.

Interval Probabilistic Logic Programs: other semantics. Lukasiewicz [17–19] and Lakshmanan and Sadri [15,16] have studied alternative sematnics for interval probabilistic logic programs.

The work of Lukasiewicz originated, in part, in the desire to simplify the semantical framework of Ng and Subrahmanian [22], while extending the functionality of the logic programming formalism. Instead of possible world semantics, Lukasiewicz in [17,18] extends the semantics of multi-valued logic to reasoning with probability intervals. In fact, [18] discusses a probabilistic disjunctive logic programming formalism. The key difference between [18] and this work lies in the semantics ascribed to probabilistic rules in the program and the means of manipulation of interval probabilities. In [18], probabilities of conjunction and disjunction are computed using the *meet* (min) and *join* (max) operations on the probability lattice described in [17]. By comparison, [21] and this work, specify the semantics of conjunction and disjunction via a linear optimization problem which follows from the possible-world semantics.

In [15,16], Lakshmanan and Sadri represented uncertainty as a pair of intervals measuring *confidence* and *doubt* independently, associated with the *probabilistic clause*, rather than with formulas in the clause. The key similarity of this framework with GAP-based approaches (including our work) is in the treatment of the $\Longleftarrow$ symbol in the clauses as a *reversed modus ponens*. The key difference between the semantics of [16] and that of [21] and this paper is in how the probability intervals for the same formulas derived from different rules are combined. The posible world semantics considered in [21] and in this paper obliges the model of a (d)p-program P to satisfy every rule in the program. At the same time, the framework for [16] allows for the models of their program to satisfy some, but not all clauses.

Bayesian approaches to logic programming. As discussed in Section 1, Bayesian approaches to logic programming treat $\Longleftarrow$ as an indicator of conditional probability. First such formalisms are due to Poole [27] and Ngo and Haddawy [24]. Both proposed frameworks are equivalent to Bayesian networks [26] in expressive power.

Baral, Gelfond and Rushton [2] create an elegant framework (P-log) which combines answer set programming [10] and bayesian reasoning. In their framework, (point) probabilities are associated with some of the propositional atoms of the langauge. The rule base is constructed using A-Prolog syntax [28], and does not include uncertainty. Each truth assignment to the propositional atoms yields has a probability of occurring: the answer sets of a P-log program are all the assignments (sets) with non-zero probability.

While previously mentioned Bayesian frameworks work with point probabilities, Lukasiewicz has also studied the use of interval probabilities in the context of bayesian inference in logic programming [19]. Just like [22] and this work, [19] uses possible world semantics, however, it is adapted for a logic programming formalism with distinctly different properties.

Conclusions. This work provides the description of the possible-world semantics for a disjunctive probabilistic logic programming langauge which naturally extends the Interval Probabilistic Satisfiability [11]. The results presented in this paper highlight the price one has to pay for reasoning with interval probabilities. On one hand, interval probabilities is the simplest and the most easy-to-understand form of imprecise probabilities. On the other hand, the results presented in this paper show that reasoning formalisms combining possible world semantics and interval probabilities *are not closed*. Dp-programs, described in the paper use only probability intervals to specify the conditions which probability assignments to propositional formulas must obey. However, in this framework, a single interval is not a precise description of the set of all possible probability assignments to a propositional formula. We hope that these observations lead to the study of more complex forms of imprecise probabilities and their incorporation into logic programming frameworks.

References

1. G. Boole. (1854) *The Laws of Thought*, Macmillan, London.
2. Chitta Baral, Michael Gelfond, J. Nelson Rushton. (2004) Probabilistic Reasoning With Answer Sets, in *Proc. LPNMR-2004*, pp. 21-33.
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. V.Chvátal. (1983) Linear Programming. *W. Freeman and Co.*, San Fancisco, CA.
5. A. Dekhtyar. (2000) Reasoning with Uncertainty and Time, *Ph. D. Thesis*, University of Maryland, College Parge, August 2000.
6. A. Dekhtyar, M.I. Dekhtyar. (2004) Possible Worlds Semantics for Probabilistic Logic Programs, in *Proc., International Conference on Logic Programming (ICLP)'2004, LNCS*, Vol. 3132, pp. 137-148.
7. A. Dekhtyar, M.I. Dekhtyar. (2005) Revisiting the Semantics of Interval Probabilistic Logic Programs, in *Proc. 8th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'05), LNAI*, Vol. 3662, pp. 330-342.
8. A. Dekhtyar and V.S. Subrahmanian (2000) Hybrid Probabilistic Programs. *Journal of Logic Programming*, Volume 43, Issue 3, pp. 187 – 250 .

9. R. Fagin J. Halpern, and N. Megiddo. (1990) A logic for reasoning about probabilities, *Information and Computation*, vol. 87, no. 1,2, pp. 78-128.
10. M. Gelfond, V. Lifschitz (1988) The Stable Model Semantics for Logic Programming. In *Proceedings, ICLP/SLP 1988*, pp. 1070–1080
11. G.Georgakopoulos, D. Kavvadias, C.H. Papadimitriou. (1988) Probabilistic Satisfiability, *Journal of Complexity*, Vol. 4, pp. 1-11.
12. T. Hailperin. (1965) Best Possible Inequalities for the Probability of a Logical Function of Events, *American Mathematical Monthly*, Vol. 72, pp. 343–359.
13. M. Kifer, V.S. Subrahmanian (1992) Theory of Generalized Annotated Logic Programming and its Applications, *Journal of Logic Programming*, Vol. 12, No. 4, pp. 335–368.
14. 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.
15. L.V.S. Lakshmanan, F. Sadri. (1994) Modeling Uncertainty in Deductive Databases. In *Proceedings, DEXA'94*, September 1994, Athens, Grees, LNCS, Vol 856, pp. 724–733, Springer.
16. L. V. S. Lakshmanan, F. Sadri. (1994) Probabilistic Deductive Databases. in *Proc. International Symposium on Logic Programming (SLP)*, pp. 254-268
17. T. Lukasiewicz. (1998) Probabilistic Logic Programming. In *Proceedings 13th European Confrence on Artificial Intelligence (ECAI'98)*, pp. 388–392, J. Wiley & Sons.
18. T. Lukasiewicz. (1999) Many-Valued Disjunctive Logic Programs with Probabilistic Semantics. In *Proceedings, 5th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR)*, LNAI, Vol. 1730, pp. 277-289, Springer.
19. T. Lukasiewicz. (2001) Probabilistic Logic Programming under Inheritance with Overriding. In *Proceedings, UAI 2001*, pp. 329-336.
20. J. Minker (1982) On Indefinite Data Bases and the Closed World Assumption. In Loveland, D., ed.: *Proceedings 6th Conference on Automated Deduction* (CADE '82). Volume 138 of Lecture Notes in Computer Science., New York, Springer, pp. 292-308.
21. R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, *Information and Computation*, 101, 2, pps 150–201, 1993.
22. 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.
23. R. Ng and V.S. Subrahmanian. (1995) *Stable Semantics for Probabilistic Deductive Databases*, INFORMATION AND COMPUTATION, 110, 1, pps 42-83.
24. L. Ngo, P. Haddawy (1995) Probabilistic Logic Programming and Bayesian Networks, in *Proc. ASIAN-1995*, pp. 286-300.
25. N. Nilsson. (1986) *Probabilistic Logic*, AI Journal 28, pp 71–87.
26. J. Pearl. (1988) *Probabilistic Reasoning in Intelligent Systems: Netoworks of Plausible Inference*, Morgan Kaufmann Publishers, 1988.
27. D. Poole (1993). Probabilistic Horn Abduction and Bayesian Networks. *Artificial Intelligence*, Vol. 64(1), pp. 81-129.
28. P. Simons, I. Niemelä, T. Soininen (2002) Extending and Implementing the Stable Model Semantics. *Artificial Intelligence Journal*, Vol. 138, pp. 181–234.
29. Walley, P. (1991). Statistical Reasoning with Imprecise Probabilities. Chapman and Hall, 1991.