The Theory of Interval Probabilistic Logic Programs

Alex Dekhtyar1*, Michael I. Dekhtyar2**

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 computationally reasoning about probabilities with 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), a disjunctive extension of a logic programming language originally considered by Ng and Subrahmanian[14,15]. 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 construction of reasoning formalisms for imprecise probabilities and suggests that interval probabilities alone are inadequate to support such reasoning.

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

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

$$A : p \leftarrow B.$$ 

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[19], Ngo and Haddawy [17], 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 propositions $A_1, \ldots, A_n$. 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 true1.

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 [11] more than a century later and has been “modernized” by Georgakopoulos, Kavvadis and Papadimitriou[10] in 1988. Nilsson’s probabilistic logic[18] is based on PSAT and uses the semantics of possible worlds (world probability functions) to model probabilities of events. In [10] 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 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.

---

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].
years [20,3]. It has led to the establishment of the Imprecise Probabilities Project [12] and has generated an interest in interval probabilistic logic programming frameworks [14,8,13].

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 \( \ldots \) 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[14,15] 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] showed that the semantics of interval probabilistic logic programming languages based on Interval PSAT, in particular, the languages considered by Ng and Subrahmanian in [14,15] is more complex. When possible world semantics of Nilsson [18] 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 [15]².

This paper extends our prior results to a larger class of programs: disjunctive interval probabilistic programs (dp-programs): interval probabilistic programs where heads of rules can contain disjunctions of formulas. An example of rule in our logic programming language is given below:

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

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 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 checking whether a dp-program has a model is NP-complete.

² Without the variable annotations for probabilities.
- **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 a 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 [14,15].

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 [15].

### 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 [14,15] with constant probability annotations. In present work, we extend the class of programs we study to the case of disjunctive logic programs.

Let \( L \) be some first order language containing infinitely many variable symbols, finitely many predicate symbols and no function symbols. Let \( B_L = \{ A_1, \ldots, A_N \} \) be the Herbrand base of \( L \). 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 \((B_1' \lor \ldots \lor B_m') : \mu'\), 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.

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, \( A_1 \lor \ldots \lor A_n : [l, u] \) is read as "the

---

\(^3\) Extended naturally to incorporate disjunctive rules.
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

\[
G_1 : \nu_1 \lor \cdots \lor G_k : \nu_k \leftarrow F_1 : \mu_1 \land \cdots \land F_n : \mu_n.
\]

where \( k \geq 0 \) and \( n \geq 0 \).

The head of the clause is its left side \( G_1 : \nu_1 \lor \cdots \lor G_k : \nu_k \) and the body of the clause is its right side \( F_1 : \mu_1 \land \cdots \land F_n : \mu_n \).

If \( k = 1 \), the \( p \)-clause \( G_1 : \nu_1 \leftarrow F_1 : \mu_1 \land \cdots \land F_n : \mu_n \) is non-disjointive. Our earlier work \([7]\) considered only such clauses, as did \([14, 15]\).

If \( k > 0, n = 0 \), as usual, the \( p \)-clause \( G_1 : \nu_1 \lor \cdots \lor G_k : \nu_k \leftarrow \) is referred to as a fact.

If \( k = 0, n > 0 \), as usual, the \( p \)-clause \( \leftarrow F_1 : \mu_1 \land \cdots \land F_n : \mu_n \) is referred to as an integrity constraint.

If \( k = 0 \) and \( n = 0 \) then empty clause \( \leftarrow \) 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-disjointive is called a non-disjointive \( p \)-program or simply a \( p \)-program.

A (d)\( p \)-program in which all basic formulas in all clauses are atoms from \( BL \) 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 \( BL \) is called a factored (d)\( p \)-program.

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

| smoke: [0.2, 0.4] | smoke: [1.1], (hot \& dry): [0.0]. |
| alarm: [0.3, 0.6] | smoke: [0.6, 1], (hot \& dry): [0.4, 0.7]. |
| alarm: [0.7, 0.9] | smoke: [0.6, 1], (hot \& dry): [0.8, 1]. |
| fire: [0.4, 1] \lor robbery: [0.1, 0.2] | alarm: [0.4, 0.7], smoke: [0.8, 1]. |
| (robbery \lor intrusion) : [0.6, 1] | alarm: [0.8, 1], smoke: [0.3], (hot \& dry): [0.2, 0.7]. |

Informally, we can think of \( P \) describing a portion of a knowledge base about the work of a home alarm system. The first \( p \)-clause of \( P \) is a fact that 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 triggering with the probability of hot and dry weather is judged to be 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 the fire is present 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 the fire is present. 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 [14] Ng and Subrahmanian considered factored p-programs. In [16] they considered a framework, in which variables were allowed in the probability annotations. 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
\[ WP : 2^{B_L} \rightarrow [0, 1], \]
\[ \sum_{W \subseteq B_L} WP(W) = 1. \]

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 \( A \in W \);

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

\( W \models A_1 \lor \ldots \lor A_n \) 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 domain is 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 straightforward 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 \( I^a : B_L \rightarrow [0, 1] \).

A probabilistic interpretation (p-interpretation) \( I \) is a function \( I : \text{bf}(B_L) \rightarrow [0, 1] \).

**Definition 6** Let \( WP \) be a world probability density function. A p-interpretation \( I_{WP} \) is defined on 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:

\[ WP(\emptyset) = p_1 = 0.2 \quad WP(\{a, b\}) = p_4 = 0.3 \]
\[ WP(\{a\}) = p_2 = 0.4 \quad WP(\{b\}) = p_3 = 0.1 \]

Then, \( WP \) induces the following p-interpretation \( I = I_{WP} \):

\[ I(a) = p_2 + p_1 = 0.7; \]
\[ I(b) = p_3 + p_4 = 0.4; \]
\[ I(a \land b) = p_4 = 0.3; \]
\[ I(a \lor b) = p_2 + p_3 + p_4 = 0.8 \]

**Example 3** Consider the following p-interpretation \( I \): \( I(a) = 1 \), \( I(b) = 1 \), \( I(a \land b) = 0 \), \( I(a \lor b) = 1 \). 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:
\[ I(a) = p_2 + p_4 = 1; \]
\[ I(b) = p_3 + p_4 = 1; \]
\[ I(a \land b) = p_4 = 0; \]
\[ I(a \lor b) = p_2 + p_3 + p_4 = 1, \]

where 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 \( p_i \)'s 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 corresponding 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 \( I : bf(B_L) \rightarrow [0, 1] \) 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 the 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 \( bf(B_L) \).

- \( I \models F_1 : \nu_1 \iff I(F_1) \in \mu_1; \)
- \( I \models F_1 \land \ldots \land F_n : \nu_n \iff (\forall 1 \leq i \leq n)(I \models F_i : \nu_i); \)
- \( I \models G_1 \lor \ldots \lor G_k : \nu_k \iff (\exists 1 \leq i \leq k)(I \models G_i : \nu_i); \)
- \( I \models G_1 \lor \ldots \lor G_k : \nu_k \leftrightarrow F_1 : \mu_1 \land \ldots \land F_n : \mu_n \iff \) either \( I \models G_1 \lor \ldots \lor G_k : \nu_k \) or \( I \not\models F_1 : \mu_1 \land \ldots \land F_n : \mu_n. \)
In particular, 
\[ I \models F_1 : \mu_1 \land \ldots \land F_n : \mu_n \iff I \not\models F_1 : \mu_1 \land \ldots \land F_n : \mu_n, \]
and always \( I \not\models \).

**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 \( WMod(P) \neq \emptyset \)), 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 \( M = 2^N \)-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 \( CONS-P = \{ P \mid Mod(P) \neq \emptyset \} \).

**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 \( EQ(P) \):

- \(- \sum_{j=1}^{M} p_j = b_i \), 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]. Similar statement is also found in [10]).
**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 \( O(r l + r \log r) \).

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 \( B_i (i = 1, \ldots, r) \) it’s probability \( b_i \in [0, 1] \) of the length \( O(r l + r \log r) \).
2) Guess a probability distribution \( WP \) with no more than \( r + 1 \) positive probabilities \( p_1, \ldots, p_{r+1} \) of the length \( O(r l + r \log r) \) 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 \( 3 \mbox{-CNF} \leq_p \mbox{CONS-P} \). Let \( \Phi = C_1 \land \ldots \land C_m \) be a 3-CNF over the set of boolean variables \( \mbox{Var} = \{x_1, \ldots, x_n\} \). Let each clause \( C_j, j = 1, \ldots, m, \) include 3 literals \( l^1_j, l^2_j, l^3_j \). Define for each literal \( l \) an annotated atom \( \alpha(l) \) as follows: if \( l = x \in \mbox{Var} \) then \( \alpha(l) = x : [0.5, 1] \), if \( l = \neg x \) then \( \alpha(l) = x : [0, 0.5] \). Let \( B_L = \mbox{Var} \cup \{C_j | j = 1, \ldots, m\} \cup \{\Phi\} \). We include in \( p \)-program \( P(\Phi) \) the following p-clauses. (f1) : \( \Phi : [1.1] \leftarrow . \)

(f2) : \( C_j : [0, 0.1] \leftarrow . \quad (j = 1, \ldots, m) \)

(f3) : \( x_i : [0, 1] \leftarrow . \quad (i = 1, \ldots, n) \)

(r1) : \( \alpha(l^1_j) \land \alpha(l^2_j) \land \alpha(l^3_j) \). \( (j = 1, \ldots, m) \).

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

**Proposition 2** \( \Phi \in 3 \mbox{-CNF} \iff P(\Phi) \in \mbox{CONS-P} \).

**Proof.** Suppose that \( \Phi \in 3 \mbox{-CNF} \) and \( \sigma : \mbox{Var} \rightarrow \{T, F\} \) is such truth substitution that \( \sigma(\Phi) = T \). Then for each \( i = 1, \ldots, m \) there is such \( k_j, 1 \leq k_j \leq 3 \), that \( \sigma(l^k_j) = T \). Define an ap-interpretation \( I^\sigma \) as follows: \( I^\sigma(\Phi) = 1, I^\sigma(C_j) = 0 \) for \( j = 1, \ldots, m, I(x_i) = 0 \) if \( \sigma(x_i) = T \) and \( I^\sigma(x_i) = 1 \) if \( \sigma(x_i) = F, i = 1, \ldots, n \). Then it is easy to see that all facts \((f1), (f2), (f3), (r1)\) and all rules \((r1)\) are valid on \( I^\sigma \). Consider now any rule \((rc_j)\). Its body includes the annotated atom \( \alpha(l^k_j) \). If \( l^k_j = x \in \mbox{Var} \) then \( \alpha(l^k_j) = x : [0.5, 1] \). By the choice of \( l^k_j \) 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 \( I(x) \neq 0.5 \) for \( i = 1, \ldots, n \). For \( x \in \text{Var} \) 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 \( x \in \text{Var} \) 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 \( x \in \text{Var} \) 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 p-program \( P \) entails a formula \( F : [l, u] \) if for each \( I \in \text{Mod}(P) \) \( I \models F : [l, u] \). 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 \( EQ_2(P, F) = EQ(P) \cup \{ \sum_{W_j \models F} p_j > u \} \). Then it easy to see that \( P \) 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.

**Theorem 2** The entailment problem for dp-programs is co-NP-complete.

We note, in fact, that in the theorems 1 and 2 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 \( \text{Mod}(P) \) 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.

Our work on the semantics of interval p-programs started with an observation that even for simple p-programs as defined in [14], \( \text{Mod}(P) \) is a complex object. 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 \( \text{Mod}(P) \) 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 $\text{Mod}(P)$ 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 $C[0, 1]$ denote the set of all subintervals of the interval $[0, 1]$. An atomic function is a mapping $f : B_L \rightarrow C[0, 1]$.

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

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

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.

**Definition 13** Given a basic formula $F$,

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

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

The $T_P$ operator is defined as follows:

$$T_P(\mu)(F) = [l_F, u_F],$$

where $l_F = \min \left( \sum_{W_j \models F} p_j \right)$, subject to $\mathcal{LL}(S_P(\mu))$ and $u_F = \max \left( \sum_{W_j \models F} p_j \right)$, subject to $\mathcal{LL}(S_P(\mu))$. 
Fig. 1 Sample p-program $P$, and the set of inequalities $LL(S_P)$ it induces.

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 thinning is required.

**Example 4** Consider a p-program $P$ which has p-clauses

$(a \land b) : [0.6, 0.9] \leftarrow .

(a \land c) : [0.8, 0.9] \leftarrow .

The $S_P$ operator will produce $S_P(a \land b) = [0.6, 0.9]$, $S_P(a \land c) = [0.8, 0.9]$, 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 $Prob(a \land b) = 0 \not\in [0.6, 0.9]$. 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 $F \in \text{bf}(B_L)$. $S_P(h)(a \land b) = [0.05, 0.5] \cap [0.5, 1] = [0.5, 0.5]$, $S_P(h)(a \land c) = [0.5, 0.5]$, $S_P(h)(b \land c) = [0.5, 0.5]$ and $S_P(h)(a \land b \land c) = [0.1, 0.2]$. To compute $T_P(h)$ we first construct $LL(S_P(h))$. Let $W_1 = \{a, b, c\}$, $W_2 = \{a, b\}$, $W_3 = \{a, c\}$ and $W_4 = \{b, c\}$. The set of inequalities $LL(S_P(h))$ is shown in Figure 1 (for simplicity we replace constraints of the form $a \leq X \leq a$ with $X = a$).

Combining the first three constraints with the fifth we get $2p_1 - 0.5 = p_5 + p_6 + p_7 + p_8$ or $p_1 = 0.25 + p_6 + p_7 + p_8$. Because all $p_i \geq 0$, $\text{min}(p_1)$ 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, $LL(h)(S_P)$ 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 $S_P(h)(a \land b \land c) = [0.1, 1]$. Now, $T_P(h)(a \land b \land c)$ is defined: $\text{min}(p_1)$ subject to $LL(S_P)(h)$ is 0.25 (see previous example for derivation), $\text{max}(p_1) = 0.5$ and it is reached when $p_2 = p_3 = p_4 = 0$. Thus, $T_P(h)(a \land b \land c) = [0.25, 0.5]$.

From [14] we know that the set of all formula functions over $\text{bf}(B_L)$ forms a complete lattice $\mathcal{F}$ w.r.t. the subset inclusion: $h_1 \leq h_2$ iff $\forall F \in
\[
\begin{array}{c|c|c}
\text{Program } P_1 & \text{Program } P_2 \\
\hline
a: [0.2, 0.4] & a: [0.2, 0.4] \\
b: [0.2, 0.3] & b: [0.2, 0.3] \\
\end{array}
\]

**Fig. 2** Sample P-programs.

\[bf(B_L))(h_1(F) \supseteq h_2(F))\]. The bottom element \(\bot\) 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 \([14]\) w.r.t. \(\mathcal{F^F}\).

The iterations of \(T^P\) are defined in a standard way:

1. \(T^P_0 = \bot\);
2. \(T^P_{n+1} = T^P(T^P_n)\), where \(\alpha + 1\) is the successor ordinal whose immediate predecessor is \(\alpha\);
3. \(T^P_\lambda = \bigcup\{T^P_n \mid n \leq \lambda\}\), where \(\lambda\) 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 \(\mathcal{L}\mathcal{L}(h)\) 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 ([14], Theorem 2). They also show that if a \(p\)-program \(P\) is consistent, then \(I(lfp(T^P))\), the set of all \(p\)-interpretations satisfying \(lfp(T^P)\) contains \(Mod(P)\) ([14, 15] 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 3** There exists a \(p\)-interpretation \(I\) s. t. \(I \in I(lfp(T^P_1))\) 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, \(T^P_1(a) = [0.2, 0.4]\) and \(T^P_1(b) = [0.2, 0.5]\) 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, \(lfp(T^P_1) = T^P_1\).

Now, consider \(p\)-interpretation \(I\), such that \(I(a) = 0.2\) and \(I(b) = 0.35\). Clearly, \(I \in I(lfp(T^P_1))\). 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, \(I(b) = 0.35 \not\in [0.4, 0.5]\), 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 4** 1. If $\text{lf}(T_{P_2}) = T_{P_2}$. In particular, $\text{lf}(T_{P_2})(a) = [0.2, 0.4]$ and $\text{lf}(T_{P_2})(b) = [0.3, 0.5]$.

2. $\text{Mod}(P_2) = \emptyset$

**Proof.** The first part is similar to the proof of Proposition 3. To show that $\text{Mod}(P_2) = \emptyset$, consider some $p$-interpretation $I$ such that $I \models P_2$.

Let $I(a) = p$. As $p \in \text{lf}(T_{P_2})(a) = [0.2, 0.4]$ 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 $I(b) \in \text{lf}(T_{P_2})(b) = [0.3, 0.5]$, which leads to a contradiction. 

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

**Proposition 5** If the atoms in $B_L$ for $P_1$ (Figure 2) are ordered as $a, b$, then $\text{Mod}(P_1) = [0.2, 0.3] \times [0.2, 0.3] \cup (0.3, 0.4] \times [0.4, 0.5]$.

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

Consider an $ap$-interpretation $I$ such that $I \models P_1$. As $\text{lf}(T_{P_1})(A) = [0.2, 0.4]$ (by rule (1)), three cases are possible.

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

2. $I(A) = 0.3$. In this case, the bodies of both rule (3) and rule (4) are satisfied, and therefore $I$ must satisfy both heads of these rules, i.e., $I(B) \in [0.2, 0.3]$ and $I(B) \in [0.4, 0.5]$. But as $[0.2, 0.3] \cap [0.4, 0.5] = \emptyset$, we arrive to a contradiction. Therefore, for any $p$-interpretation $I \models P_1$, $I(A) \neq 0.3$.

3. $I(A) \in (0.3, 0.4]$. Here, the body of rule (3) will not be true, but the body of rule (4) will, therefore, $I$ must satisfy the head of rule (4), i.e., $I(B) \in [0.4, 0.5]$. Then, $I \in (0.3, 0.4] \times [0.4, 0.5]$.

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

Looking at the proofs of the propositions above we see that the reason for the “bad” behavior of $\text{lf}(T_P)$ lies in the computation of the $SP$ operator, namely, in the determination when $p$-clauses fire. By definition of $SP$, 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 \( C : F : \mu \leftarrow G : \mu' \) 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 \( M_G = \{ \nu_j \mid G_1 : \nu_1 \vee \ldots \vee G_j : \nu_j \vee \ldots \vee G_k : \nu_k \leftarrow F_i : \mu_1 \wedge \ldots \wedge F_n : \mu_n \in P, G_i = G, (\forall 1 \leq r \leq k) \text{ if } r \neq j \text{ then } h(G_r) \cap \nu_r = \emptyset \} \) 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 into 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 ([14,15]) 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 \( Mod(P) \subseteq I(lfp(T_P)) \).

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

\[
\begin{align*}
C_1 &= a : [0,3,0.5] \lor b : [0,6,0.8] \leftarrow .
\end{align*}
\]

\[
\begin{align*}
C_2 &= a : [0,5,0.7] \lor b : [0,8,0.9] \leftarrow c : [0,3,0.5].
\end{align*}
\]

\[
\begin{align*}
C_3 &= a : [0,1,0.4] \leftarrow .
\end{align*}
\]

\[
\begin{align*}
C_4 &= c : [0,2,0.4] \leftarrow .
\end{align*}
\]

Then the first application of \( T_P (= S_P) \) to the bottom \( \bot \) 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 \( T^2_P(a) = [0,1,0.4], T^2_P(b) = [0,8,0.9] \) and \( T^2_P(c) = [0,2,0.4] \). Clause \( C_1 \) is never fired, so fixpoint \( lfp(T_P) = T^2_P(\bot) = [0,1,0.4] \times [0,8,0.9] \times [0,2,0.4] \).

Let \( I^n \) be a model of \( P \). Since \( I^n \models C_1 \) then \( I^n(a) \in [0,3,0.5] \cap [0,1,0.4] = [0,3,0.4] \) or \( I^n(b) \in [0,6,0.8] \cap [0,8,0.9] = [0,8,0.8] \). 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 $\text{Mod}(P)$ for simple p-programs and p-programs respectively. In this paper, we extend the class of programs for which similar mathematical description of $\text{Mod}(P)$ is possible to dp-programs.

4 Possible Worlds Semantics

We ask ourselves: given a p-program $P$, how do we give an exact description of $\text{Mod}(P)$? 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 $\text{Mod}(P)$.

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 $\text{Mod}(P)$. Recall, that $\text{fix}(T_P)$ 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 $M$-dimensional parallelepiped $(M = |\partial(B_L)|)$. As seen from the examples in Section 3, the bodies of p-programs can be used to “pluck” points, or even intervals from $\text{Mod}(P)$ for a specific basic formula. This means that $\text{Mod}(P)$ is no longer a solid $M$-dimensional parallelepiped, rather, it can have various portions missing. Thus, formula functions are inadequate as exact descriptors of $\text{Mod}(P)$.

Let us now construct $\text{Mod}(P)$ 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 $P$: $\text{Mod}(P)$ – models in the p-interpretation space and $\text{WMod}(P)$ – models in the probability density function space. In this section, we concentrate on building $\text{WMod}(P)$ 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 $\text{INEQ}(P)$ 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
$G_1 : [L_1, U_1] \lor \ldots \lor G_k : [L_k, U_k] \leftarrow F_1 : [l_1, u_1] \land \ldots \land F_n : [l_n, u_n].$

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

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

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

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

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

Let $P = \{C_1, \ldots, C_s\}$ and $a_0 = \{\sum_{j=1}^M p_j = 1\}$. Then, $INEQ(P)$ is defined as follows:

$INEQ(P) = \{ a_0 \cup a_1 \cup \ldots \cup a_s | a_i \in INEQ(C_i), 1 \leq i \leq s\}$.

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

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 $C = \langle G_1 : [L_1, U_1] \lor \ldots \lor G_k : [L_k, U_k] \leftarrow F_1 : [l_1, u_1] \land \ldots \land F_n : [l_n, u_n] \rangle$.

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

**Proof.** $\Rightarrow$ Let $W^P \in WMod(C)$ and $P_C = \{ p_j = WP(W_j), 1 \leq j \leq M \}$. Then $I_{WP} \in Mod(C)$, i.e. $I_{WP} \models C$. By the definition 7 at least one of the following assertions holds: (i) $I_{WP} \models G_i : [L_i, U_i] \lor \ldots \lor G_k : [L_k, U_k]$ or (ii) $I_{WP} \not\models F_1 : [l_1, u_1] \land \ldots \land F_n : [l_n, u_n]$.

In the 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 $P_C \in \text{Sol}(\alpha)$.

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

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

In the case (ii) \(\alpha\) has a form of \(\{\sum_{W_j \in F_i} p_j < l_i\}\) or \(\{\sum_{W_j \in F_i} p_j > u_i\}\)

for some \(i \in [1, n]\). The sum in the left sides of the both inequalities is equal to \(WP(F_i)\). Hence we get \(WP(F_i) < l_i\) or \(WP(F_i) > u_i\). In the both cases \(WP \not\models F_i: [l_i, u_i]\). Then by the definition \(7 WP \not\models F_i: [l_i, u_i] \wedge \ldots \wedge F_i: [l_n, u_n]\) and \(WP \models C.\)

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

**Theorem 3** 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 \(P = \{p_j = WP(W_j) | j = 1, \ldots, M\} \in Sol(\alpha)\).

**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 \(P\) is a solution of \(\alpha_C.\) Consider the following system of inequalities \(\alpha = \cup_{C \in P} \alpha_C.\) Then, \(\alpha \in INEQ(P)\) by definition of the latter and \(P\) is its solution.

Conversely, let \(P\) be a solution for some \(\alpha \in INEQ(P).\) By definition of \(INEQ(P)\), there exist systems of inequalities \(\alpha_C\) for each \(C \in P\) such that \(\alpha = \cup_{C \in P} \alpha_C.\) Then for each \(\alpha_C, P \in Sol(\alpha_C).\) 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 \(r(P)\) is 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\(^4\) (in general case) polyhedron. Given a solution \(WP\) of some \(\alpha \in INEQ(P)\), \(WP\) 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.

\(^4\) Because \(p_1 + \ldots + p_M = 1\) is present in every \(\alpha \in INEQ(P)\), the dimensionality of \(Sol(\alpha)\) cannot be more than \(M - 1\).
This corollary provides an exponential, in the size of the p-program, upper bound on the number of possibly disjoint components of $\text{Mod}(P)$. 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 general case $\text{INEQ}(P)$ is a complex construct, and using it to explicitly build $\text{WMod}(P)$ is difficult. However, when we consider the class of simple dp-programs, it turns out, it is possible to use it to define $\text{Mod}(P)$ directly, rather than the $\text{WMod}(P)$, 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 $I^a : B_L \to [0,1]$ and $\text{Mod}(P) = \{I^a | I^a \models P\}$. Using this, we can simplify the definition of $\text{INEQ}(P)$ 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 $C \equiv D_1 : [L_1, U_1] \lor \ldots \lor D_k : [L_k, U_k] \leftarrow B_1 : [l_1, u_1] \land \ldots \land B_n : [l_n, u_n]$. $k \geq 0, n \geq 0, k + n > 0$, be a clause of $P$.

The family of systems of inequalities induced by $C$, denoted $\text{INEQ}(C)$ is defined as follows:
- $\text{INEQ}(C) = \text{T}(C) \cup \text{F}(C)$;
- $\text{T}(C) = \{\{L_j \leq x_{D_j} \leq U_j | 1 \leq j \leq k\}\}$ (if $k=0$ then $\text{T}(C) = \emptyset$);
- $\text{F}(C) = \{\{x_{B_i} < l_i\}|1 \leq i \leq n\} \cup \{\{x_{B_i} > u_i\}|1 \leq i \leq n\}$ (if $n=0$ then $\text{F}(C) = \emptyset$).

Let $P = \{C_1, \ldots, C_s\}$. The family $\text{INEQ}(P)$ of systems of inequalities is defined as $\text{INEQ}(P) = \{\alpha_1 \cup \ldots \cup \alpha_s | \alpha_i \in \text{INEQ}(C_i), 1 \leq i \leq s\}$.

The key difference between Definition 16 of $\text{INEQ}(P)$ 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, $\text{INEQ}(P)$ 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 $\alpha$ of such inequalities, we denote the set of its solutions as $\text{Sol}(\alpha)$. 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 that

$$\text{Sol}(\alpha) = \begin{cases} \emptyset & \text{if for some } A, l_A^\alpha > u_A^\alpha; \\ [l_{A_1}^\alpha, u_{A_1}^\alpha] \times \ldots \times [l_{A_N}^\alpha, u_{A_N}^\alpha] & \text{otherwise.} \end{cases}$$
As with the general case, the set \( \text{INEQ}(P) \) 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 \( \text{INEQ}(P) \) captures all the systems that do.

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

**Proof.** ⇒ Let \( I \models C \). Two cases need to be considered.

1. \( \Gamma^a \models D_j : [L_j, U_j] \lor B_k : [l_k, u_k] \). Then, by definition of satisfaction, \( \Gamma^a \models D_j : [L_j, U_j] \) for some \( j \in [1, k] \). This means that \( [L_j \leq \Gamma^a(D_j) \leq U_j] \) is contained in \( T(C) \) and thus \( \alpha \in \text{INEQ}(C) \). But then \( \{x_A = \Gamma^a(A) | A \in B_L \} \) is a solution of \( \alpha \) since \( x_{D_j} = \Gamma^a(D_j) \).

2. \( \Gamma^a \not\models B_j : [l_j, u_j] \land \ldots \land B_k : [l_k, u_k] \). Then, there exists such \( 1 \leq j \leq k \) that \( \Gamma^a(B_j) \not\subseteq [l_j, u_j] \), i.e., either \( \Gamma^a(B_j) < l_j \) or \( \Gamma^a(B_j) > u_j \). In the first case, \( \{x_A = \Gamma^a(A), x_i = \Gamma^a(B_i) \mid 1 \leq i \leq k \} \) is a solution of the system \( \alpha' = \{x_{B_j} \leq l_j\} \in \text{F}(C) \), and in the second case, it will be a solution of the system \( \alpha'' = \{x_{B_j} \geq u_j\} \in \text{F}(C) \) (note that both systems consist of a single inequality, limiting the value of \( x_{B_j} \) only). As \( \text{F}(C) \subseteq \text{INEQ}(C) \), the statement of the lemma is true.

⇐ Let \( \alpha \in \text{INEQ}(C) \) and let \( U = \{x_A = p_A | A \in B_L \} \) be a solution of \( \alpha \). Consider a \( \alpha \)-interpretation \( \Gamma^a \), s.t., \( \Gamma^a(A) = p_A \). We show that \( \Gamma^a \models C \).

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

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 \( \alpha \), \( L_j \leq p_A \leq U_j \). Therefore, \( \Gamma^a(D_j) \in [L_j, U_j] \) and \( \Gamma^a \models C \).

Consider now the last remaining case, namely, \( \alpha \in \text{F}(C) \). Then \( \alpha \) 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 \( \alpha \), \( x_{B_j} < l_j \) and therefore \( \Gamma^a(B_j) \not\subseteq [l_j, u_j] \). But then, \( \Gamma^a \not\models B_j : [l_j, u_j] \lor B_k : [l_k, u_k] \), which means that \( \Gamma^a \models C \).

**Theorem 4** A \( \alpha \)-interpretation \( \Gamma^a \) is a model of a simple \( dp \)-program \( P \) iff there exists a system of inequalities \( \alpha \in \text{INEQ}(P) \) such that \( X = \{x_A = \Gamma^a(A)\} \in \text{Sol}(\alpha) \).

**Proof** follows from Lemma 4. Indeed, let \( \Gamma^a \models P \). Then for each clause \( C \in P \), \( \Gamma^a \models C \). Then by Lemma 4, there exists \( \alpha_C \in \text{INEQ}(C) \) such that \( X \) is a solution of \( \alpha_C \). Consider the following system of inequalities \( \alpha = \cup_{C \in P} \alpha_C \). Then, \( \alpha \in \text{INEQ}(P) \) 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 \( \alpha \in \text{INEQ}(P) \). By definition of INEQ(P), there exist systems of inequalities \( \alpha_C \) for each \( C \in P \) such that \( \alpha = \cup_{C \in P} \alpha_C \). Then for each \( \alpha_C, X \in \text{Sol}(\alpha_C) \).
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 $\text{Mod}(P)$:

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

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 $\text{INEQ}(C)$ is bounded by $(k(P) + 2n(P))$. Therefore, $|\text{INEQ}(P)| \leq (k(P) + 2n(P))^{r(P)}$.

The solution of each system $\alpha \in \text{INEQ}(P)$ is a convex $N$-dimensional (in general case) parallelepiped.

**Corollary 4** For a simple dp-program $P$, $\text{Mod}(P)$ is a union of at most $M(P)$ (not necessarily disjoint) $N$-dimensional parallelepipeds, 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 parallelepipeds in the set $\text{Mod}(P)$.

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 $C_i = D_i^k : [L_i, U_i] \land \ldots \land D_i^k : [L_i, U_i]$ $\leftrightarrow$. Then $\text{Mod}(P)$ can be defined by the following procedure $\text{GenModFacts}$.

1. Fix the set of indices $J = \{(j_1, \ldots, j_s) \mid 1 \leq j_s \leq k, \text{ for all } r \in [1, s]\}$.
2. For each $\alpha = (j_1, \ldots, j_s) \in J$ and $A_i \in B_L$ choose the subset of indices $J_0(A_i) = \{ k \mid D_i^k = A_i \}$.
3. If $J_0(A_i) \neq \emptyset$ then let $\nu_0(A_i) = [L_i, U_i]$ where $L_i = \max \{ L_{j_s}^k \mid k \in J_0(A_i) \}$ and $U_i = \min \{ U_{j_s}^k \mid k \in J_0(A_i) \}$.
4. If for all $i \in [1, N]$ the interval $\nu_0(A_i)$ is nonempty, i.e. $L_i \leq U_i$, then let $\text{Mod}_{\alpha}(P) = \times_{i=1}^N \nu_0(A_i)$, otherwise $\text{Mod}_{\alpha}(P) = \emptyset$.
5. Return $\text{Mod}(P) = \bigcup_{\alpha \in J} \text{Mod}_{\alpha}(P)$.

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

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

**Example 8** Let $P$ consist of $s = N$ clauses of the form $C_i = A_1 : [L_i, U_i] \land \ldots \land A_N : [L_N, U_N] \leftrightarrow$ and $[L_r, U_r] = [(i - 1)/N, (2(i - 1) + 1)/2N]$ for all $i \in [1, N]$ and $r \in [1, N]$. Then for any two permutations $\alpha_1$ and $\alpha_2$ of $(1, 2, \ldots, N)$ the sets of models $\text{Mod}_{\alpha_1}(P)$ and $\text{Mod}_{\alpha_2}(P)$ do not intersect, while for each $\alpha$ which is not a permutation $\text{Mod}_{\alpha} = \emptyset$. So, $\text{Mod}(P)$ consists of $N!$ disjoint parallelepipeds.
We can show that an exponential bound on the number of disjoint parallelepipeds 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 $\{a, b_1, \ldots, b_n\}$:

\[
a : [1, 1] \leftarrow . \quad (1)
\]

\[
b_i : [0, 1] \leftarrow i = 1, \ldots, n \quad (2i)
\]

\[
a : [0, 0] \leftarrow b_i : [0.2, 0.3], i = 1, \ldots, n \quad (3i)
\]

Here, $INEQ(1)$ consists of a single equality $x_a = 1$; each of $INEQ(2i)$ includes trivial inequalities $0 \leq x_{b_i} \leq 1$, and each of $INEQ(3i)$ consists of three systems of inequalities: $\alpha_i^1 = \{0 \leq x_{b_i} < 0.2\}$, $\alpha_i^2 = \{0.3 < x_{b_i} \leq 1\}$, and $\alpha_i^3 = \{0.2 \leq x_{b_i} < 0.3; x_a = 0\}$. Since $\alpha_i^3$ is inconsistent with $INEQ(1)$, each consistent set of inequalities in $INEQ(P')$ can be represented as $\{x_a = 1\} \cup \bigcup_{i=1}^n \alpha_i^j$ for some $j_i \in \{1, 2\}, i = 1, \ldots, n$. It is easy to see that for any two different $\alpha$ and $\alpha'$ of such form in $INEQ(P')$ sets $Sol(\alpha)$ and $Sol(\alpha')$ are disjoint. So, $Mod(P')$ consists of $2^n$ disjoint $n$-dimensional parallelepipeds. 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-dimensional parallelepipeds.

The construction from previous section gives one algorithm for computing $Mod(P)$: given a program $P$ construct explicitly the set of systems of inequalities $INEQ(P)$ and then solve each system from this set. This algorithm has exponential worst case complexity in the size of the program and as 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 $Prbp(A)$ of all possible bounds of probability intervals used in $P$ for $A$ as follows $Prbp(A) = \{\langle l, -\rangle | A : [l, u] \in AT(P)[A]\} \cup \{\langle u, +\rangle | A : [l, u] \in AT(P)[A]\} \cup \{< 0, -\rangle, < 1, +\rangle\}$. 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. $\langle a, \ast \rangle < \langle b, \ast \rangle$ whenever $a < b$, and $\langle a, \ast \rangle < \langle a, + \rangle$. Consider now $Prb_P(A) = \{\beta_1 = < 0, - >, \beta_2, \ldots, \beta_m = < 1, + >\}$ where sequence $\beta_1, \ldots, \beta_m$ is in ascending order. Using the set $Prb_P(A)$ we will now construct the set of segments $SEG_P(A)$ as follows.

Let $\beta_i = \langle a_i, \lambda_i \rangle$ and $\beta_{i+1} = \langle a_{i+1}, \lambda_{i+1} \rangle$, $1 \leq i \leq m - 1$. We define the segment $s_i$ associated with the pair $\beta_i, \beta_{i+1}$ as shown in the following table.

<table>
<thead>
<tr>
<th>$\lambda_i$</th>
<th>$\lambda_{i+1}$</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>$-$</td>
<td>$[a_i, a_{i+1})$</td>
</tr>
<tr>
<td>$-$</td>
<td>$+$</td>
<td>$[a_i, a_{i+1}]$</td>
</tr>
<tr>
<td>$+$</td>
<td>$-$</td>
<td>$(a_i, a_{i+1}]$</td>
</tr>
<tr>
<td>$+$</td>
<td>$+$</td>
<td>$(a_i, a_{i+1}]$</td>
</tr>
</tbody>
</table>

Now, $SEG_P(A) = \{s_1, s_2, \ldots, s_{m-1}\}$. Notice that if $a_i = a_{i+1}$ then, $\lambda_i$ is a “−” and $\lambda_{i+1}$ is a “+” (it follows from our order on $\beta(s)$ and the interval $[a_i, a_{i+1}] = [a_i, a_i]$ will be added to $SEG_P(A)$. The following proposition establishes basic properties of the segment sets.

**Proposition 6** 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 $A: \mu \in AT(P)[A]$, $I_1 \models A: \mu$ if and only if $I_2 \models A: \mu$.

**Proof** (1) The definition of segmentation $SEG_P(A)$ insures 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 a closed begin of the form $[a_{i+1}, a_{i+2}]$ or $(a_{i+1}, a_{i+2})$. 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 a closed interval with an open begin of the form $(a_{i+1}, a_{i+2})$ or $(a_{i+1}, a_{i+2})$. Therefore any 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 such 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 such 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 5**

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

**Proof** follows directly from proposition 6.

**Example 10** Consider again program \(P_1\) shown on Figure 2:

\[a : [0,2,0.4] \quad \text{and} \quad b : [0,2,0.5]\]

\(a\) has the set of probability bounds \(Prb_{P_1}(a) = \{(0,-), (0.2,-), (0.3,-), (0.3, +), (0.4, +), (1, +)\}\) and atom \(b\) has the set of bounds \(Prb_{P_1}(b) = \{(0,-), (0.2,-), (0.3, +), (0.4, +), (0.5, +), (1, +)\}\).

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]\} \text{ 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 \(s^1 \in SEG_{P_1}(a)\) 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 5 only 2 of them consist of models of \(P_3\): \(Mod(P_1) = [0.2,0.3] \times [0.2,0.3] \cup (0.3,0.3] \times [0.4,0.5]\).

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

**Fig. 3** Algorithm GenMod for computing \(Mod(P)\).

Theorem 5 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 \(u^i\), then for each \(i\) the value \(I(A_i)\) on step (3) can be chosen to be equal to \((l^i + u^i)/2\). So, the runtime of \(GenMod\) is bounded by a polynomial of the size of \(SEG(P)\). The size of \(SEG(P)\) is, in its turn, exponential
of the size of the set $B_L$ of all atoms of $P$. Of course, it can be a case when some “bricks” in $SEG(P)$ can be united into one larger “brick”, so that $Mod(P)$ is represented by a smaller number of bricks than $SEG(P)$. But the program $P_3$ shows that in the general case even minimal number of ”non-unitable” bricks in $Mod(P)$ can be exponential in $|B_L|$. Therefore, the worst case running time of algorithm $GenMod$ can not be improved.

At the same time, we can improve on $GenMod$, by being more careful at how the N-dimensional “bricks” are considered.

We fix an ordering $A_1, \ldots, A_N$ of $B_L$. Given a simple dp-program $P$, let $lfp(TP(A_i)) = s_{g_i}$ and $NS(P) = \times_{i=1}^N s_{g_i}$. From [14] 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 $\mu$ let us denote by $S \cap \mu$ the set $\{ s | s \in S$ 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 $\nu = \{ \{ \}, \} \subseteq [0, 1]$, $\{ \{ \}, \} \in \{ \{ \}, \} \}$, we denote by $\text{Reduct}(P, A : \nu)$ 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 7**

(1) Let $B_L = \{ A_1, \ldots, A_N \}$ and $1 \leq j \leq N$. Then $Mod(\text{Reduct}(P, A_j : \nu_j) \cup \{ A_j : \nu_j \leftarrow \}) = Mod(P \cup \{ A_j : \nu_j \leftarrow \}) = Mod(P) \cap [0, 1]^{j-1} \times \nu_j \times [0, 1]^{n-j}$.

(2) For each $s \in SEG_P(A)$ program $\text{Reduct}(P, A : \nu)$ 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 $s \cap \mu = \emptyset$. $\blacksquare$

We extend now the reduction of $P$ with respect to one atom $A : \nu$ to the reduction with respect to an atomic formula function $h : B_L \rightarrow C[0, 1]$. $\text{Reduct}(P, h)$ is returned by the following procedure:

(1) for each $A \in B_L$ do $P := \text{Reduct}(P, A : h(A))$;

(2) return $P$.

The following assertion follows straightforward from Proposition 7.
Algorithm GenModT($P$; program, $\{A_1, \ldots, A_N\}$ atoms)
if $P$ includes contradictory clause then return($\emptyset$)
else
    $Sol := \emptyset$;
    $S := NS(P)$; // compute fixpoint of Ng-Subrahmanian $T_P$ operator
    if $S = \emptyset$ then return($\emptyset$)
    else // if NS($P$) is not empty, proceed with computations
        $P := \text{Reduce}(P, S)$;
        $Seg := SEG(P, A_1) \cap sg_1$; // the segmentation of $A_1$ inside
        // the fixpoint of operator $T_P$
        // main loop
        for each $s = (a, b) \in Seg$ do
            $P' := \text{Reduce}(P, A_1 : s)$;
            if $P'$ is empty then $Sol := Sol \cup (s \times (\times_{i=2}^{N}[0, 1]))$
            else // find the solution for the reduct
                $RSol := \text{GenModT}(P', \{A_2, \ldots, A_N\})$;
                if $RSol \neq \emptyset$ then $Sol := Sol \cup (s \times RSol)$ end if
            end if
        end if
    end if
return $Sol$;

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

**Proposition 8**

1. The result of the procedure above does not depend on the order of atoms of $B_L$ in the line (1) of the procedure.
2. Let $B_L = \{A_1, \ldots, A_N\}$ be some ordering of $B_L$. Then

$$Mod(\text{Reduce}(P, h)) \cup \bigcup_{j=1}^{N} \{A_j : h(A_j) \leftarrow .\}) = Mod(\bigcup_{j=1}^{N} \{A_j : h(A_j) \leftarrow .\}) = Mod(P \cap (h(A_1) \times \ldots \times h(A_N))).$$

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 6 Given a simple p-program $P$ and an ordering $A_1, \ldots, A_N$ of $B_L$, algorithm GenModT returns the set $\text{Mod}(P)$.

Proof follows in a straightforward way from the propositions 7 and 8.

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

The advantages of GenModT over naive implementation of GenMod are demonstrated in the example of program $P_1$ (Fig. 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 GenModT $\text{Seg} = SEG_{P_1}(a) \cap (0.2, 0.4] = \{(0.2, 0.3], (0.3, 0.3], (0.3, 0.4]\}$ and the main loop will proceed three times as follows:

1) $s = [0.2, 0.3], P' = \{b : [0.2, 0.3] \leftarrow \}, \text{Sol} = \{[0.2, 0.3] \times [0.2, 0.3]\}$;

2) $s = [0.3, 0.3], P' = \{b : [0.2, 0.3] \leftarrow \}, \text{Sol} = \{[0.2, 0.3] \times [0.2, 0.3]\}$;

3) $s = [0.3, 0.4], P' = \{b : [0.4, 0.5] \leftarrow \}, \text{Sol} = \{[0.3, 0.4] \times [0.4, 0.5]\}$.

The result will be $\text{Sol} = [0.2, 0.3] \times [0.2, 0.3] \cup [0.3, 0.4] \times [0.4, 0.5]$ which is equal to $\text{Mod}(P_1)$ (see Proposition 5). Thus, 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 $\text{Mod}(P)$ exist. In particular, we ask ourselves a question of when $\text{Mod}(P)$, as defined here, and $\text{I}(\text{f}(P))$, as defined for p-programs in [15] and for simple dp-programs in Section 3 coincide. We then address the problem of complexity of detecting that $\text{Mod}(P) = \text{I}(\text{f}(P))$.

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

To simplify 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 $\text{Mod}(C) = E^N$. 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 of $A : \mu_1, \ldots, A : \mu_r$ 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 $\text{Reduct}(P, A : \nu)$ of simple dp-program $P$ w.r.t. atom $A : \nu$ was defined (see Definition 18). We strengthen this reduction and define a new reduction, $\text{Reduct}1(P, A : \nu)$ by adding to the transformations (i)-(iv) of $\text{Reduct}(P, A : \nu)$ 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 $\text{Reduct}1(P, h)$ of simple dp-program $P$ w.r.t. atomic formula function $h$ is defined in the same way as $\text{Reduct}(P, h)$.

New transformations allow sometimes 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 9**

(1) Let $B_L = \{A_1, \ldots, A_N\}$ and $1 \leq j \leq N$. Then $\text{Mod}(\text{Reduct}1(P, A_j : \nu_j) \cup \{A_j : \nu_j \leftarrow \}) = \text{Mod}(P \cup \{A_j : \nu_j \leftarrow \}) = \text{Mod}(P) \cap [0, 1]^{j-1} \times \nu_j \times [0, 1]^{N-j}$.

(2) Let $B_L = \{A_1, \ldots, A_N\}$ be some ordering of $B_L$. Then $\text{Mod}(\text{Reduct}1(P, h) \cup \bigcup_{j=1}^N \{A_j : h(A_j) \leftarrow \}) = \text{Mod}(P \cup \bigcup_{j=1}^N \{A_j : h(A_j) \leftarrow \}) = \text{Mod}(P) \cap \text{Reduct}1(P, h(A_1) \times \ldots \times h(A_N))$.

We use $\text{Reduct}1(P, h)$ to define the subclass of strict simple dp-programs.

**Definition 19** A simple dp-program $P$ is called strict if $\text{Reduct}1(P, \text{lfp}(T_P)) = \emptyset$.

For p-programs strictness can be defined as follows.

**Definition 20** Let $P$ be a p-program and let $P'$ be the result of removing from $P$ all p-clauses whose heads are satisfied by $\text{lfp}(T_P)$. A p-program $P$ is called strict if the following condition holds: for each clause $C : F : \mu \leftarrow F_1 : \mu_1 \land \ldots \land F_n : \mu_n$ in $P'$, there exists an index $1 \leq i \leq n$, such that $\text{lfp}(T_P)(F_i) \cap \mu_i = \emptyset$. 
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 7**

(1) For a simple dp-program $P$ checking whether it is strict can be performed in polynomial time.

(2) For a simple dp-program $P$, $\text{Mod}(P) = \mathcal{I}(lfp(T_P)) \iff P$ is strict.

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

(2) Suppose that $\text{Reduct1}(P,lfp(T_P)) = \emptyset$. Then from this equality and from proposition 9 (2) it follows that $\text{Mod}(\text{Reduct1}(P,lfp(T_P))) \cup \bigcup_{j=1}^{N} \{A_j : lfp(T_P)(A_j) \leftarrow \} = (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N)) = \text{Mod}(P) \cap (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N))$. Therefore, $\text{Mod}(P) = (lfp(T_P)(A_1) \times \ldots \times lfp(T_P)(A_N)) = \mathcal{I}(lfp(T_P))$.

Now suppose that $\text{Reduct1}(P,lfp(T_P)) \neq \emptyset$. Then $\text{Reduct1}(P,lfp(T_P))$ includes some clause $C$ of the form $D_1 : \nu_1 \lor \ldots \lor D_k : \nu_k$ where $D_i$'s and $\nu_i$'s are atoms in $P$. From the definition of $\text{Reduct1}(P,lfp(T_P))$ 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 $lfp(T_P)(A) = [L, U]$. (1) If $A : [l_1, u_1], A : [l_2, u_2], l_1 \leq u_1 < l_2 \leq u_2$, are two occurrences of $A$ into the head of $C$ and $A : [l, u]$ the occurrence of $A$ into the body of $C$ then put $I^a(A) = (u_1 + l_2)/2$. From the conditions on the considered intervals it follows that $I^a(A) \in [L, U] \cap [l, u]$ and $I^a(A) \not\in [l_1, u_1] \cup [l_2, u_2]$. Hence, $I^a(A) \in lfp(T_P)(A), I^a(A) = A : [l_1, u_1]$, and $I^a(A) \neq A : [l_1, u_1] \lor A : [l_2, u_2]$. (2) If $A : [l_1, u_1]$ is the only occurrence of $A$ into the head of $C$ and $A : [l, u]$ is the occurrence of $A$ into the body of $C$ then the points (v) and (vi) of the definition of $\text{Reduct1}(P,A : [L, U])$ ensure that $([L, U] \cap [l, u]) \not\subseteq [l_1, u_1]$, so we can choose a point $I^a(A)$ into 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 $A$ into 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 $\alpha_j$. (4) If $A : [l, u] = B_i : \mu_i$ is the only occurrence of $A$ into 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 $\beta_j$. (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 $I^a = B_i : \mu_i$ for all $1 \leq i \leq n$. On the other hand, for every $1 \leq j \leq k$ the value $I^a(D_j) \neq \nu_j$, hence $I^a \neq D_j : \nu_j$. Therefore, $I^a \neq C$. On the other hand, $I^a \in \mathcal{I}(lfp(T_P))$. Hence, $\mathcal{I}(lfp(T_P)) \neq \text{Mod}(P)$.

From Theorem 7 we can obtain the following corollary.
A survey of different approaches to probabilistic logic programming can be found in [8] and [9]. Logic for reasoning with probabilistic data is described in [10] and [11]. A good introduction to imprecise probabilities is the book by Walley [12]. The possible worlds semantics for imprecise probabilities can be found in [13].

This paper studies the precise semantics of probabilistic logic programming language for reasoning about the interval probabilities of events and their combinations. This language, proposed by Ng and Shahrilman [14], is a natural extension of interval propositional logic, in which all rules have no bodies.

The language considered here is a natural extension of the syntax of Ng and Shahrilman to allow for disjunctions in the heads of clauses. We show that this relatively simple language, the class of satisfying models for general probability assignments, can be found in [15].

Theorem 8. If P is a program, then Mod(P) = Mod(P).

Proof. We know that Mod(P) = Mod(P).

A corollary to this result is that if P is a program, then Mod(P) = Mod(P).

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

Example 11. Consider the following program P:
\[ a: \{0.06, 0.15\} \]
\[ b: \{0.06, 0.07\} \]
\[ c: \{0.06, 0.07\} \]
\[ d: \{0.02, 0.03\} \]
\[ e: \{0.03, 0.04\} \]
\[ f: \{0.05, 0.06\} \]
\[ g: \{0.05, 0.06\} \]

Then \( I(P) \neq \emptyset \) implies \( I(P) \neq \emptyset \) and therefore \( P \) is not strict. However, there exists no p-interpretation \( I \) which satisfies the first three rules and the body of the fourth rule. \( I(b) \in [0.3, 0.6] \) implies \( I(b) \in [0.3, 0.6] \) and \( I(c) \in [0.3, 0.5] \) implies that \( I(c) \geq 0.5 \).

Therefore, Mod(P) = Mod(P).
References