Probabilistic Logic Programming under the L-Stable Semantics

Denis Deratani Mauá[1,*], Fabio G. Cozman[3] and Alexandro Garces[4]

[1]Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil
[2]Escola Politécnica, University of São Paulo, São Paulo, Brazil
[4]MIT, Cambridge, MA, USA

Abstract

Probabilistic logic programming extends logic programming to offer a rich specification language for statistical relational models and, more recently, to neurosymbolic reasoners. The standard stable model semantics adopted by probabilistic logic programs collapses in the presence of contradictions that can arise when knowledge is not carefully elicited. In this work, we study probabilistic disjunctive logic programs under the least-undefined stable model semantics. We prove missing complexity results for logic inference with bounded-arity predicates, and then derive the complexity of probabilistic inference with respect to both the credal and the maximum entropy semantics.

Keywords

Probabilistic Logic Programming, Inconsistent Knowledge Bases, Computational Complexity, Stable Model Semantics

1. Introduction

The construction of a knowledge base may produce contradictions, for instance, when several experts are consulted, or when rules are extracted automatically [1]. A contradiction may render a knowledge base completely useless, depending on the adopted semantics. Take the following well-known logic program example that describes John and the barber as two male adults, and expresses that a barber shaves every male adult who does not shave himself:

$$\mathsf{maleAdult}(\mathsf{john}). \quad \mathsf{maleAdult}(\mathsf{barber}).$$
$$\mathsf{shaves}(\mathsf{barber}, X) \leftarrow \mathsf{maleAdult}(X), \mathbf{not}\ \mathsf{shaves}(X, X).$$

There is no stable model here, for if the barber shaves himself then $\mathsf{shaves}(\mathsf{barber}, \mathsf{barber})$ is not enforced by the rule; and if the barber does not shave himself, then $\mathsf{shaves}(\mathsf{barber}, \mathsf{barber})$ must be true. Yet the rule produces the sensible inference $\mathsf{shaves}(\mathsf{barber}, \mathsf{john})$.

We can handle such contradictions gracefully through a three-valued semantics where facts may be left *undefined* [2]. The least-undefined (partial) stable model semantics (L-stable, for short) is one such semantics [3, 4]. The semantics considers that facts can be true, false or undefined; the latter can be interpreted as denoting a local inconsistency of the knowledge base, or simply some fact whose truthiness is impossible to determine given the available state of knowledge. For example, under the L-stable semantics, a minimal model for the previous logic program considers that $\mathsf{shaves}(\mathsf{barber}, \mathsf{john})$ is true while $\mathsf{shaves}(\mathsf{barber}, \mathsf{barber})$ is undefined.

Probabilistic logic programming languages [5] extend logic programming [6] to represent relational probabilistic models. Most such languages adopt a semantics derived from Sato's distribution semantics [7], where a basic distribution over logic programs is extended into a distribution over the intended models, from which probabilistic queries of interest can be answered. The following program, written in ProbLog's syntax, gives a taste of one such a language:

$$0.01 :: \mathsf{likes}(\mathsf{anna}, \mathsf{bob}). \quad \mathsf{likes}(\mathsf{bob}, \mathsf{carl}).$$
$$\mathsf{likes}(X, Y) \leftarrow \mathsf{likes}(X, Z), \mathsf{likes}(Z, Y).$$

The program states that Anna likes Bob with probability 0.01, that Bob likes Carl, and that liking is transitive. The distribution semantics assigns a probability distribution over two logic programs: one containing the fact $\mathsf{likes}(\mathsf{anna}, \mathsf{bob})$ and the remaining facts and rules, with probability 0.01, and another one that does not contain that fact, with probability 0.99. Thus, considering the minimal model semantics for the logic programs, the probabilistic program infers that $\mathsf{likes}(\mathsf{anna}, \mathsf{carl})$ is true with probability 0.01.

The distribution semantics is appealing for its simplicity and for separating the logic and probabilistic parts, which enables different combinations of logic and probabilistic semantics. And while such a combination can be made in a number of different forms [8, 9, 10, 11, 12], most probabilistic logic programming languages require that any induced logic program admits at least one model, thus ruling out programs containing contradictions.

Targeting the modeling of probabilistic argumentative knowledge, which often contains contradictions, Totis, Kimmig and De Raedt [13] adopted the stable semantics (which extends the minimal model semantics to handle recursive definitions that go through negations) and suggested to make all atoms undefined (inconsistent, in their terminology) whenever a contradiction occurs. The approach, which they named smProbLog, copes with the requirement of existence of an intended model at the expense of introducing an extreme case of undefinedness in the semantics.

LP^{MLN} [1] instead copes with inconsistency by renormalizing the probability mass over the induced logic programs that admit a stable model. As noted in [13], doing so violates the essential assumption of Sato's semantics that probabilistic choices are independent. That leads to probabilistic inconsistencies, as inferences draw from the program might disagree with the specified marginal probabilities specified, and creates difficulties for parametric learning from data.

Hadjichristodoulou and Warren [14] advocated using the three-valued well-founded semantics [15] for the induced logic programs, which always admit a single model. The well-founded semantics does not distinguish between undefinedness that arises from inconsistencies and those that arise from multiple possible intended models. That is, an atom might be assigned as undefined either because it is involed in a contradiction (e.g. $p \leftarrow \mathbf{not}\ p.$), or because it is defined differently in more than one model (e.g. $p \leftarrow \mathbf{not}\ q.$, $q \leftarrow \mathbf{not}\ p.$). The well-founded semantics is also more undefined: given a logic program, it assigns more undefined atoms than the corresponding L-stable semantics.

22nd International Workshop on Nonmonotonic Reasoning, November 2–4, 2024, Hanoi, Vietnam
*Corresponding author.
ddm@ime.usp.br (D. D. Mauá); fgcozman@usp.br (F. G. Cozman); agarces2@mit.edu (A. Garces)

Another path has been followed by Rocha and Cozman [16], where the L-stable semantics is applied to each induced logic program. As they also focused on probabilistic argumentation, the authors considered only normal logic programs without integrity constraints, which always admit at least one L-stable model. While the L-stable semantics is not widely adopted, it is an interesting choice in that it differentiates between situations that admit more than one stable model from situations that have no stable model (and are then identified with contradictions).

To summarize: the L-stable semantics handles inconsistencies so as to minimize the number of undefined atoms, in a way that is consistent with the distribution semantics, coincides with the widely adopted stable model semantics in the lack of contradictions, and as such is able to also represent non-determinism (i.e., multiple intended models/solutions). In addition, the L-stable semantics is also akin to typical semantics of abstract argumentation frameworks [13, 17, 16].

To make the probabilistic language applicable, one needs to derive inference algorithms that take a probabilistic logic program and a target atom and produce the probabilities of that atom being true according to the given semantics. When designing such algorithms, is useful to classify such inference tasks with respect to their computational complexity class.

In this work, we study the computational complexity of probabilistic logic programming under the L-stable semantics. We first prove some missing results about the complexity of (*non-probabilistic*) logic programs with bounded-arity predicates. More precisely, we show that marginal inference under the L-stable semantics is Σ_3^p-hard for normal programs and Σ_4^p-hard for disjunctive programs, thus climbing one step of the polynomial hierarchy with respect to the stable model semantics [18]. We then prove complexity of probabilistic logic programs under two common choices of probabilistic semantics.

The credal semantics [19, 20] considers all probability distributions that are consistent with the distribution semantics requirements and the respective L-stable models, thus providing interval-valued inferences. Using our results on the complexity of logic programs, e show that computing such inferences for normal/disjunctive probabilistic programs with bounded-arity predicates is $\text{PP}^{\Sigma_3^p}$/$\text{PP}^{\Sigma_4^p}$-complete, a result that again let us climb one level of the counting hierarchy relative to the stable model semantics complexity [21].

The maximum entropy semantics averages all probability models and produces a sharp probability value [8, 22, 13]. Despite being relatively more common than the credal semantics, its complexity is largely unexplored. In fact, this work provides the first complexity results under that semantics that we are aware of. We show that the complexity of inference under the maximum entropy semantics is upper bounded by PP^{PP}, an extremely powerful class, and lower bounded by PP^{NP}, even for propositional normal programs with no undefined atoms (i.e., under the stable model semantics), and for propositional disjunctive programs with no undefined atoms and no negation. While those results still leave us with quite a large gap in the characterization of the complexity class, it shows that the maximum entropy semantics can be least as hard as the credal semantics, if not harder. That might encourage the development of inference algorithms that use the credal semantics to approximate the maximum entropy semantics.

We also address a more practical aspect of inference under the L-stable semantics. We use a known relation [23] between models under the L-stable and stable semantics to devise an algorithm that reduces inference under the former to inference under the latter. Using the algorithm, we provide empirical evidence that L-stable semantics is more informative than smProbLog semantics.

The rest of this paper is organized as follows. We start by reviewing some basic facts about logic programming and probabilistic logic programming (Section 2). We then prove complexity results first for logic programs (Section 3) and then for probabilistic logic programs (Section 4). In Section 5, we discuss the practical inference algorithm for probabilistic programs and the empirical results comparing definedness of L-stable semantics and smProbLog. Final remarks conclude this paper (Section 6).

2. Preliminaries

We assume the reader is familiar with logic programming and refer to [24] for a gentle introduction to the topic. Thus, here we only review the basic elements that help us fix some notation and terminology.

2.1. Logic Programming

A *rule* r is an expression of the form

$$H_1 \vee \cdots \vee H_k \leftarrow B_1, \ldots, B_m, \mathbf{not}\ B_{m+1}, \ldots, \mathbf{not}\ B_n\,,$$

where H_i and B_j are atoms, $k > 0$ (note: this disallows integrity constraints) and $m, n \geq 0$. We define $\mathsf{head}(r) = \{H_1, \ldots, H_k\}$, $\mathsf{body}^+(r) = \{B_1, \ldots, B_m\}$, $\mathsf{body}^-(r) = \{B_{m+1}, \ldots, B_n\}$, and $\mathsf{body}(r) = \mathsf{body}^+(r) \cup \mathsf{body}^-(r)$. A rule is *normal* if $k = 1$. It is a *fact* if in addition $m = n = 0$. In general, facts provide input data and rules derive new data or verify properties of the input data. A *disjunctive logic program*, or simply a program, is a finite set of rules. We denote the set of facts of a program P by $\mathsf{facts}(P)$. The program is normal if and only if all rules are normal.

The semantics of a program with variables is given by the semantics of its grounding, so for the rest of this section we consider only propositional programs.

The *Herbrand base* of a program is the set formed by all ground atoms that can be built using predicate names and constants in the program. The *grounding* of a program is the propositional program obtained by grounding each rule, that is, replacing variables with constants from the Herbrand base in every consistent way. Let t, f and u denote ground atoms which do *not* occur in a program P. A three-valued interpretation I is a function from the atoms in the Herbrand base to $\mathsf{f} < \mathsf{u} < \mathsf{t}$. We extend I so that $I(x) = x$ for $x = \mathsf{t}, \mathsf{u}, \mathsf{f}$, $I(\neg A) = \mathsf{t}$ if $I(A) = \mathsf{f}$, $I(A) = \mathsf{f}$ if $I(A) = \mathsf{t}$ and $I(\neg A) = \mathsf{u}$ if $I(A) = \mathsf{u}$. We say that A is *defined* if $I(A) \neq \mathsf{u}$. We write $I^x = \{A \mid I(A) = x\}$ for $x = \mathsf{t}, \mathsf{f}, \mathsf{u}$. An interpretation I is *total* if $I^{\mathsf{u}} = \emptyset$. We define $I(\mathsf{head}(r)) = \max\{I(A) \mid A \in \mathsf{head}(r)\}$ and $I(\mathsf{body}(r))$ as the minimum of $\min\{I(A) \mid A \in \mathsf{body}^+(r)\}$ and $\min\{I(\neg A) \mid A \in \mathsf{body}^-(r)\}$. An interpretation I *satisfies* a rule r iff $I(\mathsf{head}(r)) \geq I(\mathsf{body}(r))$. Equivalently, the rule is satisfied by I if and only if:

(S1) The rule body contains a false literal, that is, if either $\mathsf{body}^+(r) \cap I^{\mathsf{f}} \neq \emptyset$ or $\mathsf{body}^-(r) \cap I^{\mathsf{t}} \neq \emptyset$; or

Table 1
Interpretations, program reducts and minimal (stable) models for the program in Example 2.

id	I	$P/I - \{a \vee b\}$	MinModels(P/I)
1	(f, f)	$a \leftarrow$ t. $b \leftarrow$ t	(t, t)
2	(f, u)	$a \leftarrow$ t. $b \leftarrow$ u	(t, u)
3	(f, t)	$a \leftarrow$ t. $b \leftarrow$ f	(t, f)
4	(u, f)	$a \leftarrow$ u. $b \leftarrow$ t.	(u, t)
5	(u, u)	$a \leftarrow$ u. $b \leftarrow$ u.	(t, u), (u, t)
6	(u, t)	$a \leftarrow$ u. $b \leftarrow$ f.	(t, f), (u, t)
7	(t, f)	$a \leftarrow$ f. $b \leftarrow$ t.	(f, t)
8	(t, u)	$a \leftarrow$ f. $b \leftarrow$ u.	(t, u), (f, t)
9	(t, t)	$a \leftarrow$ f. $b \leftarrow$ f	(t, f), (f, t)

(S2) The literals in the body are all satisfied (i.e., true if positive, false if negative), and the head has at least one atom which is true; or

(S3) Each literal in the body is either satisfied or undefined, some literal is the body is undefined, and some atom in the head is either satisfied or undefined.

The last rule is what differentiates three-valued semantics, such as the L-stable semantics, from two-valued semantics, such as the stable semantics, as it allows local inconsistency introduced by a rule to be resolved by an undefined atom in the body and/or an undefined atom in the head. Note that, by definition, a fact can only be satisfied by (S2).

Example 1. *The interpretation $I(a) =$ t and $I(b) = I(c) =$ f satisfies the rule $a \vee c \leftarrow b$, **not** a. by (S1); the interpretation $I(a) =$ f and $I(b) = I(c) =$ t satisfies that same rule by (S2); last, the valuation $I(a) =$ u, $I(b) =$ t and $I(c) =$ f satisfies that rule by (S3).*

An interpretation is a *model* of a program if it satisfies all of its rules. We define a partial order $\leq$ (reflexive, antisymmetric and transitive) of interpretations as: $I_0 \leq I_1$ if and only if $I_0(A) \leq I_1(A)$ for all A. A model I_1 is *minimal* if there is no model $I_0 \leq I_1$ such that $I_0 \neq I_1$. If I is total then it is minimal if and only if I^{t} is $\subseteq$-minimal. Note that by (S3) if $I^{\mathsf{t}} = I^{\mathsf{f}} = \emptyset$, then I is a model as it satisfies all rules (and we disallow integrity constraints). Thus, since $\leq$ is a partial order and the Herbrand base is finite, every normal program admits one or more minimal models [25]. This is contrast to complete (i.e., 2-valued) semantics, for which a normal program might have none, one or multiple minimal models.

The stability of a model I is connected to the notion of the program's *reduct* w.r.t. I, written P/I, obtained by the so-called the *modified Gelfond–Lifschitz* (mGL) transformation [25]. The transformation operates on each atom $A \in \mathsf{body}^-(r)$ in the negative body of a rule $r \in P$ and replaces it by the atom t, f or u corresponding to its semantics in I. Formally, it replaces A with: (i) t if $A \in I^{\mathsf{t}}$; or (ii) f if $A \in I^{\mathsf{t}}$; or (iii) u if $A \in I^{\mathsf{u}}$. We say that I is a *stable model* of P if I is a minimal model of P/I. This is the partial stable model semantics (P-stable) for logic programs [25]. A stable model I_1 is *least undefined* (L-stable, for short) if there is no other stable model I_0 with $I_0^{\mathsf{u}} \subset I_1^{\mathsf{u}}$. That is, I is least undefined if there is no other stable model that defines (as true or false) more atoms than it. This is the L-stable model semantics [4]. We denote the L-stable models of program P by $\mathsf{models}(P)$.

Example 2. *Table 1 lists the interpretations $I = (I(a), I(b))$, the respective program reduct and minimal models for the program*

$$a \vee b. \qquad a \leftarrow \mathbf{not}\ a. \qquad b \leftarrow \mathbf{not}\ b.$$

We see that I_6 and I_8 are the only partial stable models of the program; as they define different atoms, they are also the only L-stable models.

A normal program has 1 or more L-stable models. This is because such a program always has a well-founded model, which is also a partial stable model [4].[1] The same is not true when disjunctive heads are present, as the next example shows.

Example 3. *The following disjunctive program has no L-stable model [26]:*

$$a \vee b \vee c. \quad b \leftarrow \mathbf{not}\ a. \quad c \leftarrow \mathbf{not}\ b. \quad a \leftarrow \mathbf{not}\ c.$$

To see why the program does not have stable models, note that the three normal rules induce a contradiction on the truth-value of each atom (this can be seen as a 2-coloring of an order 3 complete graph whose nodes are the atoms). Additionally, interpreting any of a, b or c as undefined leads the others to also be interpreted as undefined (due to the cycle). But this violates the disjunction.

While joint use of disjunction and negation can create inconsistent programs, disjunction can also be used to avoid the need of contradictions to discard interpretations:

Example 4. *To see another important feature of the stable semantics, consider the following program, which describes the property of 3-colorability of a given undirected graph.*

$$\mathsf{edge}(a, b). \quad \mathsf{edge}(a, c). \quad \mathsf{edge}(b, d). \quad \mathsf{edge}(c, d).$$
$$\mathsf{eq}(r, r). \quad \mathsf{eq}(g, g). \quad \mathsf{eq}(b, b).$$
$$\mathsf{edge}(X, Y) \leftarrow \mathsf{edge}(Y, X).$$
$$\mathsf{color}(X, r) \vee \mathsf{color}(X, g) \vee \mathsf{color}(X, b).$$
$$\mathsf{fail} \leftarrow \mathsf{color}(X, C), \mathsf{color}(X, D), \mathbf{not}\ \mathsf{eq}(C, D).$$
$$\mathsf{fail} \leftarrow \mathsf{edge}(X, Y), \mathsf{color}(X, C), \mathsf{color}(Y, C).$$
$$\mathsf{color}(X, C) \leftarrow \mathsf{fail}.$$
$$\mathsf{colorable} \leftarrow \mathbf{not}\ \mathsf{fail}.$$

As the underlying graph is 3-colorable, the program admits a total stable model where fail *is false. Now consider a modification of the facts so that the graph is not 3-colorable (e.g., by adding the fact* edge(b, c)*). Then any (stable) model must assign* fail *to true and therefore must also assign* color(x, c) *to true for any node x and color c. But such a model is maximal (it assigns the maximum number of true atoms); hence it is only stable, if there is not other stable model. It follows that there is a stable model defining* colorable *as true if and only if the graph is 3-colorable.*

This type of modeling feature is called *saturation*, and is important to achieve more computation power in the representation [26]. The combination of saturation and contradictions increases the computational power of the logic programming language [27].

[1] Taken as the set of true-value atoms, the well-founded model is the intersection of all P-stable models.

2.2. Probabilistic Logic Programming

A probabilistic logic program is a (disjunctive) logic program P equipped with a function $\rho : \mathsf{facts}(P) \to [0, 1]$ that assigns a probability value to each (ground) fact of the program. We follow ProbLog's syntax and write $p :: F$ to denote that $\rho(F) = p$. We adopt the common requirement that no two atoms in probabilistic facts unify with each another nor with the the head of any rule. Such a requirement is not necessary for the complexity results we prove here, but it simplifies some definitions and seems reasonable in realistic use cases.

Example 5. *We can specify a random graph of order 3 as:*

$$0.5 :: \mathsf{arc}(a, b). \quad 0.5 :: \mathsf{arc}(a, c). \quad 0.5 :: \mathsf{arc}(b, c).$$
$$\mathsf{edge}(X, Y) \leftarrow \mathsf{arc}(Y, X).$$
$$\mathsf{edge}(X, Y) \leftarrow \mathsf{arc}(X, Y).$$

The semantics of a probabilistic program is given by its grounding. Thus, we consider only grounded programs in the rest of this section. A *total choice* is any subset of the facts. Given a total choice $C \subseteq \mathsf{facts}(P)$, we denote the subsequent (deterministic) logic program by $P^C = (P \setminus \mathsf{facts}(P)) \cup C$. A probabilistic program induces a distribution over logic programs by

$$\Pr(P^C) = \prod_{F \in C} \rho(F) \prod_{F \in \mathsf{facts}(P) \setminus C} (1 - \rho(F)) .$$

That is, the logic program assumes that each fact $F \in \mathsf{facts}(P)$ is selected independently with probability $\rho(F)$ in the resulting random logic programs. Note that because we assumed that probabilistic facts do not unify with rule heads, and because under minimal model semantics an atom is true only if it appears as a fact or in the head of rule, then an atom in a probabilistic fact is true iff it is selected by a total choice. That is, a total choice is equivalent to fixing the interpretation of a probabilistic atom.

Let $I \sim C$ denote an interpretation I that agrees with C, i.e., $I^{\mathsf{t}} \cap \mathsf{facts}(P) = C$ and $I^{\mathsf{f}} \cap \mathsf{facts}(P) = \mathsf{facts}(P) \setminus C$. The probabilistic semantics can be extended to interpretations of the remaining atoms of the Herbrand base by probabilistic models. A *probabilistic model* is a probability measure over the interpretations such that

(PM1) $\Pr(I) > 0$ only if $I \in \mathsf{models}(P^C)$ for $C \sim I$, and

(PM2) $\Pr(\{I \sim C\}) = \Pr(P^C)$.

Thus, if for some total choice C the induced logic program has no L-stable model, then by (PM1) $\Pr(I) = 0$ for any $I \sim C$, which contradicts (PM2). We call such a program *inconsistent*.

The probabilistic model semantics is only defined for consistent programs, and a consistent probabilistic program admits one or more probabilistic models. When there are many probabilistic models, there are generally two ways of attributing a semantics. The *credal semantics* [12] takes the entire set of probability models, and derives tight bounds on inferences with respect to that set. It is therefore a conservative approach to probabilistic reasoning. Another choice, adopted e.g. by P-log [8], NeurASP [22] and SMProblog [13], is selecting the (single) maximum entropy (*MaxEnt*) distribution, which amounts to: $\Pr(I) = \Pr(P^C)/N^C$, where $N^C = |\mathsf{models}(P^C)|$ and $C \sim I$. That is, to uniformly select a model in $\mathsf{models}(P^C)$. That is the *MaxEnt* (for Maximum Entropy) semantics.

Table 2
Total choices and L-stable models for the program in Example 6.

	C	$\Pr(P^C)$	$\mathsf{models}(P^C)$
1	$\emptyset$	0.63	(f, f, u, t), (f, f, t, u)
2	a	0.07	(t, f, t, u)
3	b	0.27	(f, t, u, t)
4	a, b	0.03	(t, t, t, t)

Given a probabilistic model, a target atom A and a list of evidence literals $E = \{E_1, \ldots, E_n\}$, all in the Herbrand base of the program, we define a probabilistic inference as the computation of the conditional probability $\Pr(A|E)$, where $\Pr(L_1, \ldots, L_n) = \sum_{I \models L_1, \ldots, L_n} \Pr(I)$ for any list of literals L_i. For semantics that admit more than one probabilistic model, we are interested in obtaining tight upper and lower bounds on such probabilities:

$$\underline{\Pr}(A|E) = \min_{\Pr} \Pr(A|E), \quad \overline{\Pr}(A|E) = \max_{\Pr} \Pr(A|E).$$

For semantics which produce a single probability value, a conditional probability can be obtained as the ratio of two joint probability events, $\Pr(A, E)$ and $\Pr(E)$, so that there is no loss in focusing on the computation of the marginal probability of an atom. This is not the case for multivalued semantics such as the credal semantics. However, due to its structure, the credal semantics do admit a similar decomposition [20]:

$$\underline{\Pr}(A|E) = \frac{\underline{\Pr}(A, E)}{\underline{\Pr}(A, E) + \overline{\Pr}(\neg A, E)}.$$

A similar expression can be obtained for $\underline{\Pr}(A|E)$, noting that $\overline{\Pr}(A|E) = 1 - \underline{\Pr}(\neg A|E)$. Additionally, the marginal lower and upper probabilities according to the credal semantics satisfy:

$$\underline{\Pr}(A) = \sum_{C : \forall I \sim C, I \models A} \Pr(P^C),$$
$$\overline{\Pr}(A) = \sum_{C : \exists I \sim C, I \models A} \Pr(P^C).$$

That is, the lower probability collects the probabilities of total choices that induce a logic program of which A is a cautious consequence, while the upper probability collects the probabilities of logic programs of which A is a brave consequence. Credal semantics thus extends the cautious/brave reasoning strategies of answer set programming [24].

Example 6. *Consider the probabilistic program:*

$$0.1 :: a. \quad 0.3 :: b.$$
$$c \leftarrow a. \qquad d \leftarrow b.$$
$$c \vee d. \qquad c \leftarrow \mathbf{not}\ c. \qquad d \leftarrow \mathbf{not}\ d.$$

The total choices, their distribution, and the L-stable models of the induced programs (denoted as tuples $(I(a), I(b), I(c), I(d))$ are listed in Table 2. The credal semantics yields $\Pr(c) \in [0.1, 0.73]$ and the MaxEnt semantics yields $\Pr(c) = 0.415$. Also, we have that $\Pr(\neg c) = 0$, which implies that credal and MaxEnt semantics coincide for that inference, and also that the probability that c is undefined is $1 - 0.415 = 0.585$, under the MaxEnt semantics. This value can be taken as a measure of inconsistency of the program w.r.t. atom c.

2.3. Computational Complexity

Computational complexity classifies computational problems according to the type of resources and machines needed to solve them. This section collects basic definitions of concepts related to computational complexity theory. We point the reader to [28] and [29] for gentle introductions of the topic.

A language is a set of strings, and a complexity class is a set of languages. For some complexity class $\mathcal{C}$, a decision problem $\mathcal{D}$ accepts or rejects strings and can be viewed as a language. Then $\mathcal{D}$ is $\mathcal{C}$-hard if every problem $\mathcal{D}'$ in the complexity class $\mathcal{C}$ can be (many-one) reduced in polynomial time to $\mathcal{D}$ (that is, there is an algorithm that takes the input to $\mathcal{D}'$, modifies it with polynomial effort, calls $\mathcal{D}$ with the modified input, and then accepts or rejects $\mathcal{D}$). If $\mathcal{D}$ is in $\mathcal{C}$ and is $\mathcal{C}$-hard, then $\mathcal{D}$ is $\mathcal{C}$-complete. An oracle Turing machine $\mathbf{M}^{\mathcal{L}}$ is a Turing machine with additional tapes, such that it can write a string l to a tape and obtain from the oracle, in another tape and in unit time, the decision as to whether $l \in \mathcal{L}$ or not. If a class of languages $\mathcal{C}$ is defined by a set of Turing machines $\mathbf{M}$ (that is, the languages are decided by these machines), then define $\mathcal{C}^{\mathcal{L}}$ to consist of the languages defined by oracle machines in $\{\mathbf{M}^{\mathcal{L}} : \mathbf{M} \in \mathcal{M}\}$. If $\mathcal{C}$ and $\mathcal{C}'$ are sets of languages, $\mathcal{C}^{\mathcal{C}'} = \cup_{\mathcal{L}\in\mathcal{C}'}\mathcal{C}^{\mathcal{L}}$. We use well-known complexity classes P, NP, coNP. The Polynomial Hierarchy [30] includes classes such as $\Sigma^{\mathsf{P}}_{\mathsf{k}} = \mathrm{NP}^{\Sigma^{\mathsf{P}}_{\mathsf{k}-1}} = \mathrm{NP}^{\Pi^{\mathsf{P}}_{\mathsf{k}-1}}$, $\Pi^{\mathsf{P}}_{\mathsf{k}} = \mathrm{coNP}^{\Sigma^{\mathsf{P}}_{\mathsf{k}-1}} = \mathrm{coNP}^{\Pi^{\mathsf{P}}_{\mathsf{k}-1}}$ and $\Delta^{\mathsf{P}}_{\mathsf{k}} = \mathrm{P}^{\Sigma^{\mathsf{P}}_{\mathsf{k}-1}} = \mathrm{P}^{\Pi^{\mathsf{P}}_{\mathsf{k}-1}}$, We also use the complexity class PP: a language $\mathcal{L}$ is in PP when there is a nondeterministic Turing machine $\mathbf{M}$ such that $l \in \mathcal{L}$ iff more than half of computation paths of $\mathbf{M}$ accept l. The Polynomial Counting Hierarchy [31, 32] is the collection of complexity classes that includes P and such that if $\mathcal{C}$ is in the hierarchy then so are the classes of decision problems computed by oracle machines $\mathrm{PP}^{\mathcal{C}}$, $\mathrm{NP}^{\mathcal{C}}$ and $\mathrm{coNP}^{\mathcal{C}}$. The Polynomial Counting Hierarchy therefore contains the Polynomial Hierarchy, which includes classes such as $\Sigma^p_k = \mathrm{NP}^{\Sigma^p_{k-1}}$ and $\Pi^p_k = \mathrm{coNP}^{\Sigma^p_{k-1}}$ for $k > 0$, with $\Sigma^p_0 = \Pi^p_0 = \mathrm{P}$, and also counting classes with oracles in the polynomial hierarchy, such as $\mathrm{PP}^{\Sigma^p_k}$.

When discussing complexity results in the following, we consider always the decision versions of the inference problems, e.g. deciding whether $\min$ / $\max \Pr(A) \geq \gamma$ for some rational γ and atom A.

3. Complexity of Logic Programs

The first contribution of this work is to provide some missing results about the computational complexity of inference in (deterministic) logic programs under the L-stable semantics.

The complexity of logic programs under the total stable model semantics has been thoroughly analyzed in the literature, both in terms of the expressivity of the language (i.e., the presence of negation, disjunction, aggregation, etc) and the program structure (i.e., its dependency graph) [6, 26, 18]. In short, the only island of tractability is for propositional and stratified programs. The inclusion of negation makes inference coNP-hard, and allowing disjunction climbs one level in the polynomial hierarchy to Σ^p_2-hard (even if negation is disallowed). Allowing variables (but bounding predicate arity) climbs another step in the hierarchy (to Σ^p_2-hard in normal programs to Σ^p_3 hard in disjunctive programs). And allowing variables and no bound in predicates takes the problem to EXPTIME. There are also results regarding the complexity of partial stable models. Eiter and Gottlob [26] showed that deciding if there is a partial stable model for a disjunctive logic program is Σ^p_2-complete. In fact, it was shown that one can, in polynomial-time, map stable models of a program to partial stable models of an equivalent program [26] and vice-versa [23]. Thus, the ability to model undefinedness of truth-values does not seem important for computation complexity. That result does not extend to the L-stable model semantics, as partial stable models might not be least undefined.

Eiter, Leone and Saccà [27] proved that deciding if a propositional normal program has an L-stable model satisfying a given atom is Σ^p_2-complete, which is the same complexity of propositional disjunctive programs under the stable model semantics [26]. Note that inference in propositional normal programs under the P-stable model semantics is coNP-hard (i.e., Σ^p_1-hard). Hence, the L-stable semantics adds another layer of computational power. Combining negation, disjunction and minimal undefinedness increases computation power: inference in such programs is Σ^p_3-complete [27].

The complexity of non-propositional logic programs under the L-stable semantics has been open. The next results fill that gap. We start with the complexity of normal logic programs with bounded-arity predicates.

Theorem 1. *Deciding if there is an L-stable model satisfying a given atom for a normal program with bounded-arity predicates is Σ^p_3-hard.*

Proof. Hardness is shown by a many-one reduction from the canonical Σ^3_p-complete problem of deciding satisfiability of a 3-Quantified Boolean Formula (3-QBF) [31]:

$$\exists X_1, \ldots, X_m \, \forall Y_1, \ldots, Y_n \, \exists Z_1, \ldots, Z_o \, \phi \,,$$

where ϕ is in 3-CNF with clauses $C_1, \ldots, C_p$.

We encode each literal over X_i and Y_j by a predicate $\mathsf{lit}(S, N, P)$, where $S \in \{x, y\}$, N is the corresponding variable subindex and P is 0 if the literal is negated (i.e., $\neg X_1$) and 1 otherwise. For any such variable, insert the rules

$$\mathsf{lit}(S, N, P) \leftarrow \mathbf{not}\ \mathsf{lit}(S, N, Q), P \neq Q.$$

That rule creates a bijection between L-stable models and configurations x, y of the variables X_i and Y_j. Insert also:

$$\mathsf{fail}(N, P) \leftarrow \mathbf{not}\ \mathsf{fail}(N, P), \mathsf{lit}(x, N, P).$$

The effect of that rule is to make any two L-stable models that correspond to different configurations of the variables X_i incomparable w.r.t. undefinedness. Then encode each clause $C_i = L_{i1} \vee L_{i2} \vee L_{i3}$ by a set of rules $\mathsf{cl}(V_{i1}, V_{i2}, V_{i3}) \leftarrow \mathsf{lit}(S, N, P)$ and facts $\mathsf{cl}(V_{i1}, V_{i2}, V_{i3})$, as follows. If L_{ij} is a literal over a variable X_k or Y_k, then add the corresponding atom $\mathsf{lit}(S, N, P)$ in the body and instantiate the logical variable V_{ij} to the respective constant x or y. If instead L_{ij} is over a Z_k variable, then select the respective logic variable V_{ij} to the constant 1 or 0, dependening of the assignment that satisfies the clause. For example, the clause $\neg X_1 \vee Z_1 \vee \neg Z_2$ is encoded as

$$\mathsf{cl}(x, Z_1, Z_2) \leftarrow \mathsf{lit}(x, 1, 0). \quad \mathsf{cl}(x, 1, Z_2). \quad \mathsf{cl}(X_1, Z_1, 0).$$

The formula ϕ is represented as the rule

$$\mathsf{phi} \leftarrow \mathsf{cl}_1(V_{11}, V_{12}, V_{13}), \ldots, \mathsf{cl}_p(V_{p1}, V_{p2}, V_{p3}),$$

where the variables V_{ik} are selected to reflect the variables in literal L_{ik} mentioned in clause C_i (if it is an X_i/Y_j variable with use the constant x/y). Consider a configuration x, y to the variables X_i and Y_j, and a corresponding t/f-interpretation $I_{x,y}$ of the groundings of $\mathsf{lit}(S, N, P)$. Then $\exists Z\, \phi(x, y)$ is true if an only if phi is satisfied by the extension of $I_{x,y}$. Finally, insert the rule $f \leftarrow \mathbf{not}\ f, \mathsf{phi}$. The effect of that rule is to encode the universal quantification of variables Y_j. To see this, fix a configuration x of the X_i variables. Suppose there is a configuration y such that $\exists Z\, \phi(x, y)$ is false. Then there is an L-stable model I_y that defines phi (as false), defines values for all groundings of $\mathsf{lit}(S, N, P)$, and defines f as false. Suppose instead that there is no such y. Then for any model I_y satisfying phi the atom f must be undefined. Hence, there is an L-stable model satisfying phi only if Φ is true. To observe the converse, note that changing the interpretation of any atom either violates some clause or breaks minimality. Thus, checking if an L-stable model satisfies phi solves the 3-QBF problem. □

We now handle the case with disjunctive rules.

Theorem 2. *Deciding if there is an L-stable model satisfying a given atom for a disjunctive program with bounded-arity predicates is Σ_4^p-hard.*

Proof. Hardness follows from a reduction from 4-QBF:

$$\exists X_1, \ldots, X_m\, \forall Y_1, \ldots, Y_n\, \exists W_1, \ldots, W_p\, \forall Z_1, \ldots, Z_o\, \phi\,.$$

We repeat the same encoding in the proof of Theorem 1, with the following differences. First, we encode $\neg\phi$ as

$$\mathsf{negphi} \leftarrow \mathsf{cl}_i(Z_{i1}, Z_{i2}, Z_{i3}),$$

where $\mathsf{cl}_i(Z_{i1}, Z_{i2}, Z_{i3})$ is the negation of the the i-th clause. We also add $\mathsf{phi} \leftarrow \mathbf{not}\ \mathsf{negphi}$, for convenience. We represent the literals relative to the W_k variables as $\mathsf{lit}(w, N, P)$, and use the saturation technique to quantify over them:

$$\mathsf{lit}(w, N, 1) \vee \mathsf{lit}(w, N, 0). \quad \mathsf{lit}(w, N, P) \leftarrow \mathsf{negphi}.$$

The existence of a configuration w for which $\phi(x, y, w, z)$ is false for some z is represented by an interpretation defining both $\mathsf{lit}(w, N, 1)$ and $\mathsf{lit}(w, N, 0)$ as true, for $N = 1, \ldots, p$. Since such an interpretation is $\subseteq$-dominated by any interpretation that assigns true to only one of the literals $\mathsf{lit}(w, N, P)$ (and false to the other), it is a model only if $\forall W \exists Z\, \neg\phi(x, y)$. The remainder of the proof is as in Theorem 1. □

4. Complexity of Probabilistic Logic Programs

If logical inference in some logic programming language belongs to complexity class C, then probabilistic inference under the credal semantics in the corresponding probabilistic logic programming language belongs to PP^{C} [21]. We thus have:

Corollary 1. *Probabilistic inference in propositional normal programs under the credal L-stable semantics predicates is $\mathrm{PP}^{\Sigma_2^p}$-complete.*

Corollary 2. *Probabilistic inference in propositional disjunctive programs under the credal L-stable semantics predicates is $\mathrm{PP}^{\Sigma_3^p}$-complete.*

Corollary 3. *Probabilistic inference in normal probabilistic programs with bounded-arity predicates under the credal L-stable semantics is $\mathrm{PP}^{\Sigma_3^p}$-complete.*

Corollary 4. *Probabilistic inference in disjunctive probabilistic programs with bounded-arity predicates under the credal L-stable semantics is $\mathrm{PP}^{\Sigma_4^p}$-complete.*

The argument however does not apply for the MaxEnt semantics. Intuitively, this is because that semantics has no inner decision problem, hence it is not clear how to relate the complexity of logical inference to the respective probabilistic inference complexity. In the rest of this section, we provide a few results regarding the complexity of MaxEnt semantics. As will become clear, such a semantics demands more sophisticated strategies.

We start by proving complexity of the restricted case where the L-stable models in $\mathsf{models}(P^C)$ for any total choice C can be efficiently enumerated. This is the case, for example, when the number of negated atoms, the number of rules with non-empty bodies is limited or yet the size of the strongly connected components of the dependency graph are bounded [33]. Note that if we can enumerate stable models of a program, then we can also enumerate L-stable models in the same complexity [23].

Theorem 3. *Deciding whether the probability of an atom exceeds a given threshold under the MaxEnt semantics for propositional disjunctive programs is PP-complete when the L-stable models of any induced logic program are efficiently enumerated.*

Proof. Hardness follows from the complexity of stratified programs (where the number of induced stable models is 1).

Membership: We want to decide if $\Pr(A) \geq \gamma$ for some program P, atom A and rational threshold γ. For simplicity we first assume that each total choice has uniform probability and that $\gamma = 0.5$. We later discuss how to generalize those assumptions.

Guess a total choice C and let K be an upper bound on $|\mathsf{models}(P^C)|$. To be able to efficiently enumerate models, K needs to be polynomial on the input size. And since we can efficiently enumerate, we can also efficiently count both $N^C = |\mathsf{models}(P^C)|$ and the number N_A^C of such models that satisfy A. Thus guess a model in $\mathsf{models}(P^C)$ and branch the respective computation path into $K!$ paths; for $(K!/N^C) \cdot N_A^C$ of them accept the computation and for the $(K!/N^C) \cdot N_{\neg A}^C$ remaining, reject it. Let N denote the number of total choices. The machine accepts with probability

$$\frac{\sum_{C \subseteq \mathsf{facts}(P)} (K!/N^C) \cdot N_A^C}{\sum_{C \subseteq \mathsf{facts}(P)} K!} = \sum_{C \subseteq \mathsf{facts}(P)} \Pr(P^C) \frac{N_A^C}{N_C},$$

which equals $\Pr(A)$. To lift the assumption of $\gamma = 0.5$, branch any accepting path such that it accepts with probability $0.5 + (1 - \gamma)$ and branch any rejecting path such that it accepts with probability $0.5 - \gamma$. Then we accept with probability $\Pr(A) \cdot [0.5 + (1 - \gamma)] + [1 - \Pr(A)](0.5 - \gamma) = 0.5 + \Pr(A) - \gamma$, which is greater than 0.5 iff $\Pr(A) > \gamma$. To remove the assumption of uniform total choice probabilities, proceed as follows. For each atom $A \in C$ with rational probability $\rho(A) = p/q$, branch into q paths: for p of them insert A into the logic program then proceed as before; for the remaining $q - p$ do not insert A and also proceed as before. Then a total choice C is selected with

probability $\Pr(P^C)$, and the machine accepts with probability $\sum_{C \in \mathsf{facts}(P)} \Pr(P^C) N_A^C / N^C = \Pr(A)$. □

We now consider the more general case, that is, when model enumeration is not tractable.

Theorem 4. *Deciding if the probability of an atom exceeds a given threshold under the MaxEnt semantics for disjunctive programs with bounded-arity predicates is in PP$^{\text{PP}}$.*

Proof. Denote by $n = |\mathsf{atoms}(P)|$ the number of probabilistic facts and take $m = |\mathsf{atoms}(P)| - n$ (for the grounded program). Assume w.l.o.g. that the threshold γ is 0.5, and that $\Pr(P^C) = 1/2^n$ for all C. Guess a total choice C and use the PP oracle to obtain the counts N_C and N_A^C; note that $\text{PP}^{\Sigma_k^p} \subseteq \text{P}^{\text{PP}}$ for any $k > 0$ [34]. Now, let $\tilde{N}_A^C / 2^p$ be an approximation from above of N_A^C / N^C with p-bits of precision, where p does not depend on C. Branch the computation into 2^p of paths and accept $\tilde{N}_A^C$ of them, while rejecting the rest. The machine hence accepts with probability

$$\frac{\sum_C \tilde{N}_A^C}{\sum_C 2^p} = \sum_C \Pr(P^C) \frac{\tilde{N}_A^C}{2^p} \geq \sum_C \Pr(P^C) \left[\frac{N_A^C}{N^C} - \frac{1}{2^p} \right]$$

which is $\Pr(A) - 2^{-p}$. If $\Pr(A) < 0.5$, then because N_A^C and N_C are integers, we have that $\Pr(A) \leq 0.5 - 2^{-n-m}$. Thus, if we choose $p = n+m+1$, then the machine accepts iff $\Pr(A) \geq 0.5$. □

While we conjecture that inference under the MaxEnt semantics is PP$^{\text{PP}}$-complete, we have not yet been able to prove so. We can however show evidence that MaxEnt inference is at least as hard as inference under the credal semantics. Recall that, under the credal semantics, inference for propositional probabilistic normal logic programs whose stable models are all total (i.e., no atom is assigned undefined) is PP$^{\text{NP}}$-complete. The following theorem shows that for those programs, MaxEnt inference is at least as hard.

Theorem 5. *Deciding if the probability of an atom exceeds a given threshold under the MaxEnt semantics for propositional normal programs is PP$^{\text{NP}}$-hard, even if all atoms are defined.*

Proof. We reduce from PP$^{\text{NP}}$-complete problem MAJ-E-SAT, which consists of deciding if for the majority of configurations of a set of Boolean variables $X_1, \ldots, X_n$ there is a configuration of Boolean variables $Y_1, \ldots, Y_m$ that satisfy a given CNF formula $\phi(X_1, \ldots, Y_m)$. More formally, for each configuration $x = x_1, \ldots, x_n$ let $N(x)$ denote the number of configurations of $y_1, \ldots, y_m$ such that $\phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is true. The MAJ-E-SAT problem is to determine if

$$t = \left| \{x \in \{0,1\}^n : N(x) \neq 0\} \right| > 2^{n-1}.$$

Let p be the number of clauses in ϕ, and p_i be the number of literals in the i-th clause. We represent each literal in ϕ by an atom $\mathsf{lit}(v, i, s)$, where $v \in \{\mathsf{x}, \mathsf{y}\}$ indicates the set of variables to which the literal belongs, i is the respective variable index in that set and $s \in \{0, 1\}$ indicates the signal (if negated or not). We define an operator $\xi(i, j)$ that returns the atom $\mathsf{lit}(v, i, s)$ corresponding to the jth literal in the ith clause of ϕ. For example, if ϕ is $(X_1 \vee \neg X_2 \vee Y_1) \wedge (\neg X_1)$, then $\xi(1, 3) = \mathsf{lit}(\mathsf{y}, 1, 1)$ and $\xi(2, 1) = \mathsf{lit}(\mathsf{x}, 1, 0)$. Thus assemble the program:

$$0.5 :: \mathsf{lit}(\mathsf{x}, i, 1). \qquad [i = 1, \ldots, n]$$
$$\mathsf{lit}(\mathsf{x}, i, 0) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{x}, i, 1). \qquad [i = 1, \ldots, n]$$
$$\mathsf{lit}(\mathsf{y}, i, 1) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{y}, i, 0). \qquad [i = 1, \ldots, m]$$
$$\mathsf{lit}(\mathsf{y}, i, 0) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{y}, i, 1). \qquad [i = 1, \ldots, m]$$
$$\mathsf{clause}(\mathsf{i}) \leftarrow \xi(i, j). \qquad [i = 1, \ldots, p] \; [j = 1, \ldots, p_i]$$
$$\mathsf{phi} \leftarrow \mathsf{clause}(1), \ldots, \mathsf{clause}(p).$$

For example, if ϕ is $(X_1 \vee \neg X_2 \vee Y_1) \wedge (\neg X_1)$, then the assembled program is

$$0.5 :: \mathsf{lit}(\mathsf{x}, 1, 1). \quad 0.5 :: \mathsf{lit}(\mathsf{x}, 2, 1).$$
$$\mathsf{lit}(\mathsf{x}, 1, 0) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{x}, 1, 1).$$
$$\mathsf{lit}(\mathsf{x}, 2, 0) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{x}, 2, 1).$$
$$\mathsf{lit}(\mathsf{y}, 1, 1) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{y}, 1, 0).$$
$$\mathsf{lit}(\mathsf{y}, 1, 0) \leftarrow \textbf{not } \mathsf{lit}(\mathsf{y}, 1, 1).$$
$$\mathsf{clause}(1) \leftarrow \mathsf{lit}(\mathsf{x}, 1, 1). \quad \mathsf{clause}(1) \leftarrow \mathsf{lit}(\mathsf{x}, 2, 0).$$
$$\mathsf{clause}(1) \leftarrow \mathsf{lit}(\mathsf{y}, 1, 1). \quad \mathsf{clause}(2) \leftarrow \mathsf{lit}(\mathsf{x}, 1, 0).$$
$$\mathsf{phi} \leftarrow \mathsf{clause}(1), \mathsf{clause}(2).$$

Fix a total choice C, hence a semantics for $\mathsf{lit}(\mathsf{x}, 1, 0), \ldots, \mathsf{lit}(\mathsf{x}, n, 1)$. The mutual negative dependency of $\mathsf{lit}(\mathsf{y}, i, 1)$ on $\mathsf{lit}(\mathsf{y}, i, 0)$ and vice-versa, makes so that we potentially have one stable model for each of the 2^m configurations of $y_1, \ldots, y_m$. The atom phi is true in any such stable model iff ϕ is satisfied by the corresponding configuration of X_i and Y_i. Now let k be some integer that we will soon define and extend that program with the following rules:

$$\mathsf{aux}(i, 1) \leftarrow \mathsf{phi}, \textbf{not } \mathsf{aux}(i, 0). \qquad [i = 1, \ldots, k]$$
$$\mathsf{aux}(i, 0) \leftarrow \mathsf{phi}, \textbf{not } \mathsf{aux}(i, 1). \qquad [i = 1, \ldots, k]$$

The new program has 2^k stable models for each stable model of the previous program that satisfies phi, and it has one stable model for each stable model of the previous program that does not satisfy phi. Thus, each x corresponds to a total choice C, and we have that $N^C = 2^k \cdot N(x) + (2^m - N(x))$. It follows that

$$\Pr(\mathsf{phi}) = \frac{1}{2^n} \sum_{x \in 0,1^n} \frac{2^k \cdot N(x)}{2^k \cdot N(x) + \left(2^m - N(x)\right)}.$$

Recall that our intention is to use $\Pr(\mathsf{phi})$ to decide if t, the number of configurations x for which $N(x)$ is nonzero, is greater than 2^{n-1}. To this end, let $\epsilon = t - 2^n \Pr(\phi)$ be the error introduced by our reduction. Note that since t is an integer then either $t \geq 2^{n-1} + 1$ or $t \leq 2^{n-1}$. Hence, if $\epsilon < 1$, we can use the value of $\Pr(\mathsf{phi})$ to decide the MAJ-E-SAT problem. That error is maximized when $N(x) = 1$ for each of the 2^n configurations x, hence:

$$\epsilon \leq 2^n - \frac{2^{k+n}}{2^k + (2^m - 1)} < 1 \Leftrightarrow 2^k > (2^n - 1)(2^m - 1).$$

Thus, selecting $k = m + n$ ensures that $\epsilon < 1$. To conclude, if $\Pr(\phi) > 1/2$ we decide that MAJ-E-SAT is true, otherwise we decide that it is false. □

Inference under the credal semantics is also PP$^{\text{NP}}$-hard for propositional probabilistic disjunctive programs with no negation (which always admit a total stabel model). The next result shows that the same lower complexity is obtained for the MaxEnt semantics.

Theorem 6. *Deciding if the probability of an atom exceeds a given threshold under the MaxEnt semantics for propositional disjunctive programs without negation is PP^{NP}-hard.*

Proof. Again, we reduce from MAJ-E-SAT with a CNF formula ϕ. Assemble the program:

$$
\begin{aligned}
&0.5::\mathsf{lit}(\mathsf{x}, i, 1). && [i = 1, \ldots, n]\\
&\mathsf{lit}(\mathsf{x}, i, 0) \vee \mathsf{lit}(\mathsf{x}, i, 1). && [i = 1, \ldots, n]\\
&\mathsf{lit}(\mathsf{y}, i, 0) \vee \mathsf{lit}(\mathsf{y}, i, 1). && [i = 1, \ldots, m]\\
&\mathsf{clause(i)} \leftarrow \xi'(i, 1), \ldots, \xi'(i, p_i). && [i = 1, \ldots, p]\\
&\mathsf{negphi} \leftarrow \mathsf{clause}(i). && [i = 1, \ldots, p]\\
&\mathsf{lit}(\mathsf{y}, \mathsf{i}, \mathsf{0}) \leftarrow \mathsf{negphi}. && [i = 1, \ldots, n]\\
&\mathsf{lit}(\mathsf{y}, \mathsf{i}, \mathsf{1}) \leftarrow \mathsf{negphi}. && [i = 1, \ldots, n]\\
&\mathsf{phi} \vee \mathsf{negphi}.
\end{aligned}
$$

The operator $\xi'(i, j)$ above returns the atom $\mathsf{lit}(v, i, s)$ corresponding to the negation of the jth literal in the ith clause of ϕ. For example, if ϕ is $X_1 \vee \neg X_2 \vee Y_1$, then $\xi'(1, 1) = \mathsf{lit}(\mathsf{x}, 1, 0)$.

Compared to the program in the proof of Theorem 5, the program above replaces negative cycles by disjunctions, and represents the negation of ϕ by the atom negphi. It also adds rules to employ the saturation technique. The effect is as follows. For a fixed total choice C, that fixes the semantics of $\mathsf{lit}(\mathsf{x}, i, 1)$ for $i = 1, \ldots, n$, there is *not* a configuration $y_1, \ldots, y_m$ that satisfies $\phi(x, y)$ iff the (single) stable model satisfies negphi. This is because such a model is saturated: flipping the interpretation of any atom from true to false must cease being a model, else the model would not be minimal (as it assigns all atoms $\mathsf{lit}(\mathsf{y}, i, 0/1)$ to true). Hence, $2^n \Pr(\mathsf{phi})$ counts the number t of assignments x for which $N(x) > 0$. □

5. Computing Inferences

To benefit from state-of-the-art ASP solvers that adopt (total) stable model semantics, we revisit a translation developed in [23] to obtain partial stable models as (total) stable models of a translated program. Given a disjunctive logic program P, we first specify a set of atoms $\underline{a}$ for each atom a in the Herbrand base of P. The translation produces a new program $P' = Tr(P)$ such that:

$$
\begin{aligned}
P' = \{&a \leftarrow b, \mathbf{not}\ \underline{c};\ \underline{a} \leftarrow \underline{b}, \mathbf{not}\ c \mid a \leftarrow b, \mathbf{not}\ c \in P\}\\
&\cup \{\underline{a} \leftarrow a \mid a \in \mathrm{Hb}(P)\},
\end{aligned}
$$

where $\mathrm{Hb}(P)$ denotes the Herbrand Base of program P. That is, the translation splits each rule into two rules whose negative body and head belong to different sets of atoms (original or duplicated atoms). The rationale is that the three possible assignments of an atom a in the source program are represented by the possible joint assignments of a pair of atoms $(a, \underline{a})$, with one joint assignment being ruled out in any stable model.

Example 7. *Take the program in Example 2. Its translation is then the program*

$$
\begin{gathered}
a \vee b. \quad \underline{a} \vee \underline{b}.\\
a \leftarrow \mathbf{not}\ \underline{a}. \quad \underline{a} \leftarrow \mathbf{not}\ a. \quad \underline{a} \leftarrow a.\\
b \leftarrow \mathbf{not}\ \underline{b}. \quad \underline{b} \leftarrow \mathbf{not}\ b. \quad \underline{b} \leftarrow b.
\end{gathered}
$$

Table 3 shows interpretation $(a, \underline{a}, b, \underline{b})$ and the respective program reducts their minimal models. To save space, we only list interpretations that satisfy rules $P' = \{a \vee b.\ \underline{a} \vee \underline{b}.\ \underline{a} \leftarrow a.\ \underline{b} \leftarrow b\}$, as these are always in the reduct (and thus must be satisfied by any model). The interpretations in bold show the (total) stable models (which are also L-stable). The partial stable models of the original program are obtained by verifying whether atoms A and $\underline{A}$ disagree on the model.

Example 8. *Consider the following program*

$$a \leftarrow \mathbf{not}\ c. \quad c \leftarrow \mathbf{not}\ a. \quad b \leftarrow a, \mathbf{not}\ b.$$

The first two rules enforce that in any stable model we either have both a and c undefined or both defined as different values. The third rule causes b to be undefined if a is true, and false when a is false (so a can be seen as the "enabler" of the rule). Thus, the only L-stable model is $I(a) = I(b) = 0$ and $I(c) = 1$. The translation of the program is

$$
\begin{gathered}
a \leftarrow \mathbf{not}\ \underline{c}. \quad \underline{a} \leftarrow \mathbf{not}\ c.\\
c \leftarrow \mathbf{not}\ \underline{a}. \quad \underline{c} \leftarrow \mathbf{not}\ a.\\
b \leftarrow a, \mathbf{not}\ \underline{b}. \quad \underline{b} \leftarrow \underline{a}, \mathbf{not}\ b.\\
\underline{a} \leftarrow a. \quad \underline{b} \leftarrow b. \quad \underline{c} \leftarrow b.
\end{gathered}
$$

The translated program admits the following stable models $I = (I(a), I(\underline{a}), I(b), I(\underline{b}), I(c), I(\underline{c}))$:

$$(\mathsf{t}, \mathsf{t}, \mathsf{f}, \mathsf{f}, \mathsf{f}, \mathsf{t}), \quad (\mathsf{f}, \mathsf{f}, \mathsf{t}, \mathsf{t}, \mathsf{f}, \mathsf{f}), \quad (\mathsf{f}, \mathsf{t}, \mathsf{f}, \mathsf{t}, \mathsf{f}, \mathsf{t})\,.$$

The second stable model corresponds to the single total stable model of the original program.

As a proof of concept, we have implemented an inference algorithm based on the described translation in the dPASP system, available at http://github.com/kamel-usp/dpasp. Our implementation work as follows. Given a probabilistic logic program and a target atom, we obtain a new program with the inclusion of the auxiliary variables. Then we enumerate the total choices. For each total choice, we call clingo [35] to enumerate the stable models, from which we obtain the L-stable models by the mapping described. We then collect all L-stable models that satisfy the target atom and compute the desired probabilistic semantics (i.e., we average the respective probabilities, if MaxEnt is used, otherwise, we compute lower and upper bounds w.r.t. to the set of L-stable models for the credal semantics). The procedure is thus inefficient in the number of probabilistic atoms and the number of stable models of the translated program.

Example 9. *We compare inference under the L-stable semantics with inference under the smProbLog semantics, proposed in [13]. Recall that smProbLog adopts a modified stable model semantics, where all atoms are considered undefined whenever the program has no stable model. We use a reduced variant of the asthma example provided in [13]. According to the example, people that smoke develop asthma, and those that have asthma do not smoke. Additionally, people can decide to smoke for some unknown cause or be influenced to smoke by some other smoker; the presence of an asthmatic, smoking person thus leads to a contradiction. To represent a negative consequence ¬*a *(e.g., people that have asthma do not smoke), we include auxiliary atoms* a_pos *and* a_neg *that represent, respectively, positive and negative variants of* a. a_pos *replaces each occurrence of* a *in a rule's head; similarly,* a_neg *replaces occurrences of ¬*a *in a rule's head. We also add a*

Table 3
Interpretations in the format $I = (a, \underline{a}, b, \underline{b})$, program reducts and minimal (stable) models for the program in Example 7.

id	I	$P/I - P'$	MinModels(P/I)
3	$(\mathsf{f},\mathsf{f},\mathsf{t},\mathsf{t})$	$a \leftarrow \mathsf{t}.\ \underline{a} \leftarrow \mathsf{t}.\ b \leftarrow \mathsf{f}.\ \underline{b} \leftarrow \mathsf{f}.$	$(\mathsf{t},\mathsf{t},\mathsf{f},\mathsf{f})$
6	$\underline{(\mathsf{f},\mathsf{t},\mathsf{t},\mathsf{t})}$	$a \leftarrow \mathsf{f}.\ \underline{a} \leftarrow \mathsf{t}.\ b \leftarrow \mathsf{f}.\ \underline{b} \leftarrow \mathsf{f}.$	$(\mathsf{t},\mathsf{t},\mathsf{f},\mathsf{f}),\ \underline{(\mathsf{f},\mathsf{t},\mathsf{t},\mathsf{t})}$
7	$(\mathsf{t},\mathsf{t},\mathsf{f},\mathsf{f})$	$a \leftarrow \mathsf{f}.\ \underline{a} \leftarrow \mathsf{f}.\ b \leftarrow \mathsf{t}.\ \underline{b} \leftarrow \mathsf{t}.$	$(\mathsf{f},\mathsf{f},\mathsf{t},\mathsf{t})$
8	$\underline{(\mathsf{t},\mathsf{t},\mathsf{f},\mathsf{t})}$	$a \leftarrow \mathsf{f}.\ \underline{a} \leftarrow \mathsf{f}.\ b \leftarrow \mathsf{f}.\ \underline{b} \leftarrow \mathsf{t}.$	$\underline{(\mathsf{t},\mathsf{t},\mathsf{f},\mathsf{t})}, (\mathsf{f},\mathsf{f},\mathsf{t},\mathsf{t})$
9	$(\mathsf{t},\mathsf{t},\mathsf{t},\mathsf{t})$	$a \leftarrow \mathsf{f}.\ \underline{a} \leftarrow \mathsf{f}.\ b \leftarrow \mathsf{f}.\ \underline{b} \leftarrow \mathsf{f}.$	$(\mathsf{t},\mathsf{t},\mathsf{f},\mathsf{f}), (\mathsf{f},\mathsf{f},\mathsf{t},\mathsf{t})$

rule so that a *is true iff* a_pos *is true and* a_neg *is false. The complete program is described as follows:*

$$\mathsf{person}(1).\quad \mathsf{person}(2).\quad \mathsf{person}(3).\quad \mathsf{person}(4).$$
$$0.1 :: \mathsf{asthma}(X).\quad 0.3 :: \mathsf{stress}(X).$$
$$0.3 :: \mathsf{influences}(1,2).\quad 0.6 :: \mathsf{influences}(2,1).$$
$$0.2 :: \mathsf{influences}(2,3).\quad 0.7 :: \mathsf{influences}(3,4).$$
$$0.9 :: \mathsf{influences}(4,1).$$
$$0.6 :: \mathsf{inh\_stress}(X).\quad 0.6 :: \mathsf{inh\_smokes}(X).$$
$$\mathsf{smokes\_pos}(X) \leftarrow \mathsf{stress}(X), \mathbf{not}\ \mathsf{inh\_stress}(X).$$
$$\mathsf{asthma}(X) \leftarrow \mathsf{smokes}(X), \mathbf{not}\ \mathsf{inh\_smokes}(X).$$
$$\mathsf{smokes\_pos}(X) \leftarrow \mathsf{influences}(Y,X), \mathsf{smokes}(Y).$$
$$\mathsf{smokes\_neg}(X) \leftarrow \mathsf{asthma}(X).$$
$$\mathsf{smokes}(X) \leftarrow \mathsf{smokes\_pos}(X), \mathbf{not}\ \mathsf{smokes\_neg}(X).$$

As the program has only one L-stable model for each total choice, the credal and MaxEnt semantics coincide. The following table compares the probability of smokes(X) *being undefined for each person, under the L-stable and smProblog semantics.*

Semantics	$\Pr(\mathsf{smokes}(X) = \mathsf{u})$			
	1	2	3	4
smProbLog	0.2223	0.2223	0.2223	0.2223
L-stable	0.1548	0.0828	0.0599	0.0909

One notes that the smProblog semantics assigns a lower probability mass to undefined states due to the allowing only one model with undefined atoms per total choice when faced with inconsistencies, while the L-stable semantics admits several and thus sums over all possible undefined models. In contrast, the L-stable semantics differentiates the different degrees to which atoms are involved in inconsistencies.

Much more can be said about algorithms for inference under the L-stable semantics. For example, to make the procedure less inefficient, we can employ any of the many approximate strategies from the probabilistic answer set solving literature, such as searching for stable models according to the their relative probability [36], or resorting to sampling strategies [37, 38]. We leave the development and analysis of more efficient inexact methods as future work.

6. Conclusion

The L-stable semantics provides a very convenient treatment for expressing knowledge with default negation (hence nonmonotonic reasoning), nondeterminism (multiple models), and local inconsistency. The last property is particular important for probabilistic extensions of logic programming, since an inconsistent program has typically no semantics.

In this work, we have examined the complexity of probabilistic disjunctive logic programs under the L-stable semantics, thus filling a number of gaps in the literature (both for programs without and with probabilities). We showed complexity results for programs without variables and with variables and bounded-arity predicates under the credal semantics (which provides set-valued probabilistic inferences). And we partially characterized the complexity of inference under the maximum entropy semantics. Despite most literature in probabilistic logic programming with multiple models being focused on the latter, there has been virtually no complexity analysis thus far, even in the absence of contradictions.

A few extensions to those results should be relatively easy to build based on the techniques presented here. For instance, inferences that concern the search for maximum probability models or probabilistic inferences when the program is fixed (i.e., data complexity) are likely to be obtained with much similar results. We leave that as future work.

This work leaves open a seemingly much harder question, namely, how to close the gap between lower and upper complexity for inference under the maximum entropy semantics. Interestingly, the maximum entropy semantics might seem at first to lead to counting problems with obvious counting oracles; however, the "inner" counting of models does not contain itself a decision problem — hence the need for the more sophisticated proofs we had to build. We presented her an upper bound that shows that inference in that semantics is at most $\mathrm{PP}^{\mathrm{PP}}$. That complexity reduces to PP, for programs whose intended models are efficiently enumerable. And we showed that the inference is already $\mathrm{PP}^{\mathrm{NP}}$-hard in propositional programs with no contradictions and that have either no negation or no disjunction, thus matching the complexity of credal semantics [21]. We expect those results to foster future research to close the gap and fully characterize the complexity of credal semantics.

Acknowledgments

The authors of this work would like to thank the generous support from the Center for Artificial Intelligence at University of São Paulo (C4AI-USP), the São Paulo Research Foundation (FAPESP grants #2019/07665-4 and 2022/02937-9), the IBM Corporation, the Brazilian National Research Council (CNPq grants no. 305136/2022-4 and 305753/2022-3) and from CAPES (Finance Code 001).

References

[1] J. Lee, Y. Wang, Weighted rules under the stable model semantics, in: Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2016, pp. 145–154.
[2] S. Kripke, Outline of a theory of truth, The Journal of Philosophy 72 (1975) 690–716.
[3] D. Saccà, C. Zaniolo, Stable models and nondeterminism in logic programs with negation, in: Pro-

ceedings of the 9th ACM-PODS Symposium, 1990, pp. 205–218.
[4] D. Saccà, C. Zaniolo, Deterministic and nondeterministic stable models, Journal of Logic and Computation 7 (1997) 555–579.
[5] F. Riguzzi, Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning, River Publishers, 2018.
[6] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Computing Surveys 33 (2001) 374–425.
[7] T. Sato, A statistical learning method for logic programs with distribution semantics, in: Proceedings of the 12th International Conference on Logic Programming (ICLP), 1995, pp. 715–729.
[8] C. Baral, M. Gelfond, J. N. Rushton, Probabilistic reasoning with answer sets, Theory and Practice of Logic Programming 9 (2009) 57–144.
[9] L. De Raedt, A. Kimmig, H. Toivonen, Problog: A probabilistic prolog and its application in link discovery, in: Proceedings of the 20th International Joint Conference in Artificial Intelligence (IJCAI), volume 7, 2007, pp. 2462–2467.
[10] S. Michels, A. Hommersom, P. Lucas, M. Velikova, A new probabilistic constraint logic programming language based on a generalised distribution semantics, Artificial Intelligence 228 (2015) 1–44.
[11] Í. Í. Ceylan, T. Lukasiewicz, R. Peñaloza, Complexity results for probabilistic datalog±, in: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI), 2016, pp. 1414–1422.
[12] F. G. Cozman, D. D. Mauá, The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference, International Journal of Approximate Reasoning 125 (2020) 218–239.
[13] P. Totis, L. De Raedt, A. Kimmig, smproblog: Stable model semantics in problog for probabilistic argumentation, Theory and Practice of Logic Programming 23 (2023) 1198–1247. doi:10.1017/S147106842300008X.
[14] S. Hadjichristodoulou, D. S. Warren, Probabilistic logic programming with well-founded negation, in: International Symposium on Multiple-Valued Logic, 2012, pp. 232–237.
[15] A. van Gelder, K. Ross, J. S. Schlipf, The well-founded semantics for general logic programs, Journal of the ACM 38 (1991) 620–650.
[16] V. H. N. Rocha, F. G. Cozman, A credal least undefined stable semantics for probabilistic logic programs and probabilistic argumentation, in: 19th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2022, pp. 309–319.
[17] P. M. Dung, On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming, and n-person games, Artificial Intelligence 77 (1995) 321–357.
[18] T. Eiter, W. Faber, M. Fink, S. Woltran, Complexity results for answer set programming with bounded predicate arities and implications, Annals of Mathematics and Artificial Intelligence (2007) 51–123.
[19] T. Lukasiewicz, Probabilistic description logic programs, International Journal of Approximate Reasoning 45 (2007) 288–307.
[20] F. G. Cozman, D. D. Mauá, On the semantics and complexity of probabilistic logic programs, Journal of Artificial Intelligence Research 60 (2017) 221–262.
[21] D. D. Mauá, F. G. Cozman, Complexity results for probabilistic answer set programming, International Journal of Approximate Reasoning 118 (2020) 133–154.
[22] Z. Yang, A. Ishay, J. Lee, NeurASP: Embracing neural networks into answer set programming, in: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI), 2020, pp. 1755–1762.
[23] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, J.-H. You, Unfolding partiality and disjunctions in stable model semantics, ACM Transactions on Computational Logic 7 (2006) 1–37.
[24] T. Eiter, G. Ianni, T. Krennwallner, Answer Set Programming: A Primer, Springer-Verlag Berlin, 2009, pp. 40–110.
[25] T. Przymusinski, Stable semantics for disjunctive programs, New Generation Computing 9 (1991) 401–424.
[26] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Annals of Mathematics and Artificial Intelligence 15 (1995) 289–323.
[27] T. Eiter, N. Leone, D. Saccá, Expressive power and complexity of partial models for disjunctive deductive databases, Theoretical Computer Science 206 (1998) 181–218.
[28] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[29] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009.
[30] L. J. Stockmeyer, The polynomial-time hierarchy, Theoretical Computer Science 3 (1977) 1–22.
[31] K. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Informatica 23 (1986) 325–356.
[32] J. Tóran, Complexity classes defined by counting quantifiers, Journal of the ACM 38 (1991) 753–774.
[33] R. Ben-Eliyahu, A hierarchy of tractable subsets for computing stable models, Journal of Artificial Intelligence research 5 (1996).
[34] S. Toda, PP is as hard as the polynomial-time hierarchy, SIAM Journal on Computing 20 (1991) 865–877.
[35] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, CoRR abs/1705.09811 (2017).
[36] J. Pajunen, T. Janhunen, Solution enumeration by optimality in answer set programming, Theory and Practice of Logic Programming 21 (2021) 750–767.
[37] D. Azzolini, E. Bellodi, F. Riguzzi, Approximate inference in probabilistic answer set programming for statistical probabilities, in: A. Dovier, A. Montanari, A. Orlandini (Eds.), AIxIA 2022 – Advances in Artificial Intelligence, 2023, pp. 33–46.
[38] D. Tuckey, A. Russo, K. Broda, PASOCS: A parallel approximate solver for probabilistic logic programs under the credal semantics, CoRR abs/2105.10908 (2021).