International Journal of Approximate Reasoning 118 (2020) 133–154

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Complexity results for probabilistic answer set programming

Denis Deratani Mauá [a,*], Fabio Gagliardi Cozman [b]

[a] Institute of Mathematics and Statistics, Universidade de São Paulo, Brazil
[b] Escola Politécnica, Universidade de São Paulo, Brazil

ARTICLE INFO

Article history:
Received 16 May 2019
Received in revised form 25 October 2019
Accepted 9 December 2019
Available online 16 December 2019

Keywords:
Probabilistic logic programming
Answer set programming
Computational complexity

ABSTRACT

We analyze the computational complexity of probabilistic logic programming with constraints, disjunctive heads, and aggregates such as sum and max. We consider propositional programs and relational programs with bounded-arity predicates, and look at cautious reasoning (i.e., computing the smallest probability of an atom over all probability models), cautious explanation (i.e., finding an interpretation that maximizes the lower probability of evidence) and cautious maximum-a-posteriori (i.e., finding a partial interpretation for a set of atoms that maximizes their lower probability conditional on evidence) under Lukasiewicz's credal semantics.

1. Introduction

Probabilities and logic programming have been combined in a variety of ways [1–8]. A particularly interesting and powerful combination is offered by probabilistic answer set programming, which exploits the powerful knowledge representation and problem solving toolset of answer set programming [9]. Available surveys describe probabilistic logic programming in detail and go over many promising applications [10–13].

The complexity of probabilistic answer set programming has been examined for definite, normal, and disjunctive programs, both with restrictions of acyclicity and without it, and under a number of semantics [14–16]. Yet, the analysis is far from complete; for example, it lacks constructs such as aggregates, and special cases such as positive disjunctive programs — some are relatively simple, others are more intricate.

In this work, we continue the study of the complexity of probabilistic disjunctive programs with integrity constraints and aggregates, all of them under Lukasiewicz's "credal semantics" [4]. This semantics specifies a set of probability distributions over the answer sets of derived logic programs, and coincides with Sato's popular distribution semantics [8,17] for programs with a single answer set (e.g. negation-stratified nondisjunctive programs). We close most of the open questions for three common types of inference: cautious reasoning (CR), which asks for the minimum probability assigned by some probability model for a target atom, most probable explanations (MPE), which asks for the most conservative interpretation consistent with a given a set of literals, and maximum a posteriori (MAP) inference, which asks for the most conservative interpretation of a selected set of predicates consistent with a given set of literals. In deriving some of these results we prove complexity results for *non-probabilistic* relational programs with aggregates and bounded-arity predicates, a topic that was left open in the literature.

The complexity results for CR and MPE parameterized by the type of constructs allowed (negation, disjunction, stratified aggregate, etc.) are summarized in Table 1. Unless otherwise indicated, each cell in this table indicates completeness under

* Corresponding author.
E-mail address: ddm@ime.usp.br (D.D. Mauá).

https://doi.org/10.1016/j.ijar.2019.12.003
0888-613X/

many-one reductions. One can discern interesting patterns from our results. One of them is the fact that complexity of probabilistic programs typically mirrors the complexity of the corresponding logical problems except for an added "base" machine (a PP "base" for CR and a NP "base" for MPE). Another remarkable pattern that can be found in our results is the difference between sum and max aggregates when programs have bounded-arity predicates; while in propositional programs the different aggregates have the same effect, in relational programs the sum aggregates require a "top" PP oracle, while max aggregates require a "top" NP oracle.

The complexity analysis for MAP is much more regular. The problem is $\mathsf{NP}^{\mathsf{PP}}$-complete for propositional programs for all languages we consider, and also for bounded-arity programs without aggregates. As with the previous inferences, the use of sum aggregates and variables introduces an extra complexity, as we show that the problem becomes $\mathsf{NP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete, an intriguing result that does not seem to have a parallel in the logic programming literature.

These results have practical consequences in the development of efficient inference algorithms. For example, this suggests that programs containing (recursive) aggregate atoms cannot be rewritten without aggregates if an exponential blow up is to be avoided. Indeed the importance of our complexity results lies not only in clarifying where languages and inferences lie within the vast hierarchy of complexity classes, but also in suggesting algorithmic ways to approach these problems. For instance, a $\mathsf{NP}^{\mathsf{PP}}$-hard problem should be solved by some search scheme helped by a model counting method; trying a simple search technique will (likely) not do.

The paper is organized as follows. Section 2 reviews concepts from (nonprobabilistic) answer set programming. Probabilistic logic programming under the credal semantics is reviewed in Section 3. The contributions of this work appear in Section 4. Section 5 contains a summary of the paper and comments on future work.

2. Answer set programming

We first review disjunctive logic programs with aggregates under the semantics advocated by Faber, Pfeifer and Leone [18]. The presentation is a bit long as the definition of syntax and semantics is somewhat involved. Readers familiar with answer set programming may skip to Section 3.

Even though our presentation is self-contained, readers unfamiliar with logic programming might benefit from reading earlier work in the area [9,19].

2.1. Syntax

Fix a vocabulary of variables, predicates, and constants and aggregate function symbols (e.g. max, sum, count). Each predicate is associated with a nonnegative number called its *arity*. As usual, variables are represented with a capital letter, while predicates start with a lower letter. For convenience, we assume without loss of generality that the set of constants are the integers.

A **standard atom** is an expression $\mathsf{p}(t_1, \ldots, t_n)$ where p is a predicate of arity n and $t_1, \ldots, t_n$ are each either a constant or a variable. An atom is **ground** if it contains no variables. A **literal** is either a standard atom (also called positive literal) or a standard atom preceded by the special keyword not (also called a negative literal). For example, $\mathsf{edge}(X, Y)$ and $\mathsf{not\,root}(0)$ are literals; the former is positive and not ground, whereas the second is negative and ground.

A **ground set** is an expression $\mathbf{z}_1 : \mathbf{A}_1; \ldots; \mathbf{z}_n : \mathbf{A}_n$, where each $\mathbf{z}_i$ is a comma-separated list of constants (i.e., integers) and each $\mathbf{A}_i$ is a comma-separated list of standard ground atoms. A **symbolic set** is of the form $X_1, \ldots, X_n : A_1, \ldots, A_m$ where each X_1 is a variable and each A_j is a standard atom. An **aggregate atom** is an expression of the form $\#f\{S\} \circ t$, where f is an aggregate function symbol, which here we consider to be one of sum, count or max, S is either a ground set or a symbolic set, $\circ$ is one of $<, >, \leq, \geq, =, \neq$, and t is a constant or a variable. Here is a ground aggregate atom (hence containing a ground set): $\#\mathsf{sum}\{1 : \mathsf{p}(1,1); 2 : \mathsf{p}(1,2)\} > 1$. And here is an aggregate atom containing a symbolic set: $\#\mathsf{count}\{Y : \mathsf{pa}(Y, X)\} \neq 1$.

A **rule** r is an expression of the form[1]

$$H_1 \vee \cdots \vee H_m \leftarrow B_1, \ldots, B_n\,,$$

where $H_1, \ldots, H_m$ are standard atoms, and $B_1, \ldots, B_n$ are standard literals and aggregate atoms. The set of atoms H_i form the **head** of the rule (denoted as $\mathsf{head}(r)$) and the set of atoms B_j is the **body** (denoted as $\mathsf{body}(r)$). The rule is said to be **normal** if $m = 1$, **disjunctive** if $m > 1$, **positive** if it contains no negative literal, and **aggregate-free** if there are no aggregate atoms. It is a **fact** if it is normal and the body is empty. We write facts without the arrows (e.g. root instead of root $\leftarrow$) for clarity. A rule with $m = 0$ and $n > 0$ is an **integrity constraint**.

A **logic program** is a set of rules. The program is normal if all rules are normal, positive if all rules are positive and so on. The program is **definite** if it is normal, positive and aggregate-free.

[1] We do not allow for *strong negation*. One can emulate strong negation with integrity constraints and (default) negation [9]. Hence, for languages containing such features no generality is lost. We leave for the future the study of languages with strong negation without default negation or integrity constraint. We also do not allow for negated aggregate atoms, as these can be rewritten without not by rewriting the operator $\circ$ of interest.

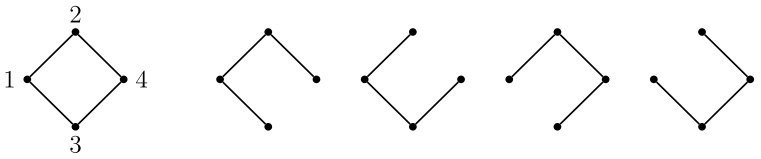

Fig. 1. The graph specified by the program in Example 1, and its corresponding spanning trees.

Example 1. The following is a logic program specifying spanning trees of the graph in Fig. 1.

$$\mathsf{root}(1).\quad \mathsf{edge}(1,2).\quad \mathsf{edge}(1,3).\quad \mathsf{edge}(2,4).\quad \mathsf{edge}(3,4). \tag{st1}$$

$$\mathsf{edge}(X,Y) \leftarrow \mathsf{edge}(Y,X). \tag{st2}$$

$$\mathsf{pa}(X,Y) \vee \mathsf{absent}(X,Y) \leftarrow \mathsf{edge}(X,Y). \tag{st3}$$

$$\leftarrow \mathsf{pa}(X,Y), \mathsf{pa}(Y,X). \tag{st4}$$

$$\leftarrow \mathsf{root}(X), \mathsf{pa}(Y,X). \tag{st5}$$

$$\leftarrow \mathsf{not\ root}(X), \#\mathsf{count}\{Y : \mathsf{pa}(Y,X)\} \neq 1. \tag{st6}$$

The relation pa defines a oriented spanning tree of that graph rooted at node 1. Each edge is either in that tree or it is absent. The atoms in rules (st1) are ground, while the remaining atoms are not. The rules in (st1) are facts; rule (st2) is definite (i.e., positive, normal and aggregate-free), rule (st3) is disjunctive and rules (st5), (st5) and (st6) are integrity constraints. Except for (st6) all rules are aggregate-free.

Given a rule containing an aggregate atom, the variables that only appear in aggregate atoms are called **local**; the remaining variables are **global**. The variable Y is local in rule (st6), and the variable X is global. Note that a variable not appearing in any aggregate atom in the rule is global. Also, a local variable need not appear in the variable list in a symbolic set. For example, Z and W are local for $\mathsf{p}(X) \leftarrow \mathsf{q}(X,Y), \#\mathsf{max}\{Z : \mathsf{p}(Z), \mathsf{q}(Y,W)\}$, and X and Y are global.

A program is **propositional** if it does not contain variables, which implies that aggregate atoms are expressed using ground sets only.

A substitution is a mapping of variables to constants in the program. The **grounding** of a rule is obtained in two steps. First a substitution of the global variables is applied. Then a propositional rule is obtained by transforming each symbolic set $\mathbf{X} : \mathbf{A}$ into the ground set $\mathbf{X}\theta_1 : \mathbf{A}\theta_1, \mathbf{X}\theta_2 : \mathbf{A}\theta_2, \ldots$, where $\theta_1, \theta_2, \ldots$ are all the substitutions for local variables in the symbolic set, and $\mathbf{X}\theta$ (resp., $\mathbf{A}\theta$) denotes the result of substitution θ to variables $\mathbf{X}$ (resp., standard atoms $\mathbf{A}$). The grounding of a program is the union of all groundings of all rules.

Example 2. Here is a positive normal logic program:

$$\mathsf{a}(X). \qquad \mathsf{b}(0,1).$$

$$\mathsf{p}(X) \leftarrow \mathsf{a}(X), \#\mathsf{sum}\{Y : \mathsf{b}(X,Y)\} < X.$$

$$\mathsf{a}(1) \leftarrow \mathsf{p}(1).$$

Grounding the program above produces the propositional program:

$$\mathsf{a}(0).\quad \mathsf{a}(1).\quad \mathsf{b}(0,1).$$

$$\mathsf{p}(0) \leftarrow \mathsf{a}(0), \#\mathsf{sum}\{0 : \mathsf{b}(0,0); 1 : \mathsf{b}(0,1)\} < 0.$$

$$\mathsf{p}(1) \leftarrow \mathsf{a}(1), \#\mathsf{sum}\{0 : \mathsf{b}(1,0); 1 : \mathsf{b}(1,1)\} < 1.$$

$$\mathsf{a}(1) \leftarrow \mathsf{p}(1).$$

A **stratification** ℓ assigns each ground standard atom of a propositional program to a nonnegative integer (called its *stratum*). A propositional program is **negation-stratified** if there exists a stratification ℓ such that for each rule r, for each atom A in the head of r, and for each standard literal B in r (possibly appearing inside of an aggregate atom), we have that [20]:

1. If B also appears in the head then $\ell(A) = \ell(B)$.
2. If B is a positive literal in the body then $\ell(A) \geq \ell(B)$.
3. If B is a negative literal in the body then $\ell(A) > \ell(B)$.

A program is negation-stratified if its grounding is negation-stratified. Intuitively, a program is negation-stratified if no atom is defined in terms of its negation through a sequence of rules. Clearly, every positive program is negation-stratified. The programs in Examples 1 and 2 are stratified.

Example 3. Here is a program which is *not* negation-stratified:

$$\mathsf{x}(0) \leftarrow \text{not } \mathsf{x}(1). \qquad \mathsf{x}(1) \leftarrow \text{not } \mathsf{x}(0). \qquad \mathsf{negphi} \leftarrow \mathsf{x}(0), \mathsf{y}(0).$$
$$\mathsf{y}(0) \leftarrow \text{not } \mathsf{y}(1). \qquad \mathsf{y}(1) \leftarrow \text{not } \mathsf{y}(0). \qquad \mathsf{negphi} \leftarrow \mathsf{z}(0).$$
$$\mathsf{z}(0) \leftarrow \text{not } \mathsf{z}(1). \qquad \mathsf{z}(1) \leftarrow \text{not } \mathsf{z}(0). \qquad \mathsf{sat} \leftarrow \text{not } \mathsf{negphi}.$$

The program encodes the Boolean satisfiability problem $\exists X, Y, Z : (X \vee Y) \wedge Z$. Atom $\mathsf{x}(0)$ represents the literal $\neg X$ in the Boolean formula, $\mathsf{x}(1)$ represents X, and so on for the predicates y and z. Atom negphi represents the negation of the Boolean formula, and sat is true iff the formula is satisfiable. Programs of this sort will be important to prove complexity results later.

A propositional program is **stratified on an aggregate atom** C if there exists a stratification ℓ such that for each rule r, for each atom A in the head of r and for each standard literal B in r, we have that [18]:

1. If B appears in the head then $\ell(A) = \ell(B)$.
2. If B is in the body then $\ell(A) \geq \ell(B)$.
3. If B appears in C then $\ell(A) > \ell(B)$.

A program is **aggregate-stratified** if its grounding is stratified on all of its (ground) aggregate atoms. Intuitively, aggregate-stratified programs do not allow recursion through aggregates, that is, no atom can be defined in terms of itself by means of an aggregate atom. The programs in Examples 1, 2 and 3 are aggregate-stratified.

Example 4. Here is a negation-stratified program which is *not* aggregate-stratified:

$$\mathsf{x}(0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(0); 1 : \mathsf{x}(1)\} \leq 0. \qquad \mathsf{negphi} \leftarrow \mathsf{x}(0), \mathsf{y}(0).$$
$$\mathsf{x}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(0); 1 : \mathsf{x}(1)\} \geq 0. \qquad \mathsf{negphi} \leftarrow \mathsf{z}(0).$$
$$\mathsf{y}(0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{y}(0); 1 : \mathsf{y}(1)\} \leq 0.$$
$$\mathsf{y}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{y}(0); 1 : \mathsf{y}(1)\} \geq 0.$$
$$\mathsf{z}(0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{z}(0); 1 : \mathsf{z}(1)\} \leq 0. \qquad \mathsf{z}(0) \leftarrow \mathsf{phi}.$$
$$\mathsf{z}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{z}(0); 1 : \mathsf{z}(1)\} \geq 0. \qquad \mathsf{z}(1) \leftarrow \mathsf{phi}.$$
$$\mathsf{phi} \leftarrow \text{not } \mathsf{negphi}.$$

The program encodes the quantified Boolean satisfiability problem $\exists X, Y \forall Z : (X \vee Y) \wedge Z$.

A program is **stratified** if it is both negation- and aggregate-stratified.

2.2. Semantics

We now establish the semantics of propositional programs. The semantics of programs with variables is the semantics of its grounding.

The **Herbrand base** of a program is the set of ground standard atoms formed by combining predicates and constants in the program. An **interpretation** is a subset of the Herbrand base. A standard atom A is true w.r.t. an interpretation I if $A \in I$, otherwise the atom is false. A negative literal not A is true w.r.t. I if $A \notin I$, otherwise it is false. We write $I \models L$ if the literal L is true w.r.t. I.

An **aggregate function** is a mapping from sets of tuples of constants (i.e., integers) to constants (integers). Each aggregate function symbol is associated with an aggregate function. Here we focus on the aggregate functions $\mathsf{count}(Z) = |Z|$, $\mathsf{sum}(Z) = \sum_{(z_1,\ldots,z_m)\in Z} z_1$ with $\mathsf{sum}(\emptyset) = 0$, and $\mathsf{max}(Z) = \mathsf{max}_{(z_1,\ldots,z_n)\in Z} z_1$ with $\mathsf{max}(\emptyset)$ undefined.

A ground aggregate atom $\#f\{S\} \circ z$ is true w.r.t. an interpretation I if Z is in the domain of f and $f(Z) \circ z$ holds, where Z is the set of tuples of constants $\{(z_1, \ldots, z_n) \mid (z_1, \ldots, z_n : A_1, \ldots, A_m) \in S, I \models A_1, \ldots, A_m\}$. A ground aggregate atom is false w.r.t. I if it is not true. For example, the interpretation $I = \{\mathsf{x}(1)\}$ satisfies the aggregate atom

$$\#\mathsf{sum}\{-1 : \mathsf{x}(0); 1 : \mathsf{x}(1)\} \geq 0.,$$

as $\#\mathsf{sum}(\{1\}) = 1 > 0$, and does not satisfy

$$\#\mathsf{sum}\{-1 : \mathsf{x}(0); 1 : \mathsf{x}(1)\} \leq 0.$$

The interpretation $I = \{\mathsf{x}(0)\}$ satisfies the latter atom and not the former.

A rule r is satisfied by an interpretation I, denoted as $I \models r$, if either some atom in the body of r is false or all the atoms in the body and some atom in the head are true.

Given a propositional program L and an interpretation I, we define the **reduct** of L w.r.t. I as the program L^I obtained by removing rules whose body contains some literal which is false w.r.t. I [18]:

$$L^I = \{r \mid r \in L, \forall B \in \mathsf{body}(r) : I \models B\}.$$

Note that we adopt the definition of reduct by Faber, Leone and Pfeifer [18], which differs from the more common Gelfond-Lifschitz reduct [21]. Either definition assigns the same semantics to programs without aggregates, and the latter is undefined for aggregates. Many other semantics have been proposed to cope with (recursive) aggregates [22–24]; for simplicity, we consider only Faber et al.'s semantics, and leave the analysis with other semantics as future work.

An interpretation I is a **model** of a program L if it satisfies all of its rules, in which case we write $I \models L$. I is an **answer set** of L if it is a subset-minimal model of its reduct, that is, if $I \models L^I$ and there is no $I' \subset I$ such that $I' \models L^I$. We denote by $\mathcal{AS}(L)$ the set of all answer sets of L.

Example 5. Consider the program L in Example 4, and the interpretations $I_1 = \{\mathsf{x}(1), \mathsf{y}(1), \mathsf{z}(0), \mathsf{z}(1), \mathsf{phi}\}$ and $I_2 = \{\mathsf{x}(1), \mathsf{y}(1), \mathsf{z}(0), \mathsf{negphi}\}$. Then L^{I_1} is

$$\mathsf{x}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(0); 1 : \mathsf{x}(1)\} \geq 0. \qquad \mathsf{negphi} \leftarrow \mathsf{z}(0).$$

$$\mathsf{y}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{y}(0); 1 : \mathsf{y}(1)\} \geq 0.$$

$$\mathsf{z}(0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{z}(0); 1 : \mathsf{z}(1)\} \leq 0. \qquad \mathsf{z}(0) \leftarrow \mathsf{phi}.$$

$$\mathsf{z}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{z}(0); 1 : \mathsf{z}(1)\} \geq 0. \qquad \mathsf{z}(1) \leftarrow \mathsf{phi}.$$

$$\mathsf{phi} \leftarrow \mathsf{not}\ \mathsf{negphi}$$

and L^{I_2} is

$$\mathsf{x}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(0); 1 : \mathsf{x}(1)\} \geq 0. \qquad \mathsf{negphi} \leftarrow \mathsf{z}(0).$$

$$\mathsf{y}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{y}(0); 1 : \mathsf{y}(1)\} \geq 0.$$

$$\mathsf{z}(1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{z}(0); 1 : \mathsf{z}(1)\} \geq 0.$$

I_1 is not a model of L^{I_1} because it does not satisfy rule $\mathsf{negphi} \leftarrow \mathsf{z}(0)$. I_2 is an answer set: removing any atom either changes the program reduct or produces the same program reduct but makes some rule unsatisfied.

2.3. Inference

The tree more common type of reasoning with logic programs are:

Satisfiability: Given a program L decide whether it has at least one answer set.

Brave Reasoning: Given a program L and a ground literal Q, decide whether *some* answer set that satisfies Q; we then say that Q is a brave consequence of L.

Cautious Reasoning: Given a program L and a ground literal Q, decide whether *all* answer sets satisfy Q; we then say that Q is a cautious consequence of L.

These inferential tasks are listed in increasing order of computational complexity. In fact, satisfiability can be cast as brave reasoning by inserting a dummy literal Q, which is a brave consequence iff the original program has some answer set; and one can perform brave reasoning by adding a rule $Q' \leftarrow \mathsf{not}\ Q$, where Q' is a fresh atom, and querying whether Q' is a cautious consequence: this is true iff Q is *not* a brave consequence [25].

3. Probabilistic logic programs

We now review the definition of probabilistic logic programs, their syntax and semantics.

3.1. Syntax

The syntax of probabilistic logic programs is a straightforward extension of the syntax of answer set programs. A **probabilistic fact** is a pair $\mu :: A$ where $\mu \in [0, 1]$ and A is a standard atom. A **probabilistic logic program** is a pair (L, F) where L is a logic program and F is a set of probabilistic facts. Intuitively, a probabilistic fact indicates that the corresponding atom may or may not be present in a logic program with some associated probability. Probabilistic logic programs therefore represent a set of a (non-probabilistic) logic programs. Since F contains only facts (annotated with probabilities), the logic program (L, F) inherits the classification of L; for example, (L, F) is normal if L is normal, disjunctive if L is disjunctive, and so on.

Example 6. Here is a stratified normal probabilistic program that counts the number of models of the Boolean formula $\phi = (X \vee \neg Y) \wedge Z$:

$$0.5 :: \mathsf{x}. \quad 0.5 :: \mathsf{y}. \quad 0.5 :: \mathsf{z}. \tag{ms1}$$

$$\mathsf{clause} \leftarrow \mathsf{x}. \quad \mathsf{clause} \leftarrow \mathsf{not\ y}. \tag{ms2}$$

$$\mathsf{phi} \leftarrow \mathsf{clause},\ \mathsf{z}. \tag{ms3}$$

The set F contains the probabilistic facts in (ms1), while the remaining rules constitute the non-probabilistic part L.

3.2. The distribution semantics

We start with the semantics of stratified normal programs, as they are much simpler to define and understand [8,13]. We consider only propositional logic programs; the semantics of programs with variables is the semantics of their groundings.

So consider a propositional probabilistic logic program (L, F). The Herbrand base of that program is the Herbrand base of $L \cup F$. A **total choice** is a subset of the Herbrand base that contains only predicates in F. We denote the set of all total choices as $\mathcal{T}(F)$. Intuitively, a total choice represents a selection of probabilistic facts that hold true, while its complement takes on false. A probabilistic logic program is **consistent** if for any total choice $C \in \mathcal{T}(F)$ the logic program $L \cup C$ has at least one answer set. One can check that the program in Example 6 is consistent. We only consider in this work probabilistic logic programs that are consistent.

The definition of a probabilistic logic program allows a fact to be associated with two different probabilities, say $0.5 :: a$ and $0.3 :: a$. While we could generalize the semantics to cope with such cases [15], we assume hereafter that no such two probabilistic facts appear in F. The complexity results we obtain later do not rely on this assumption, but it simplifies the discussion of the semantics, and does not seem to be an important modeling feature.

A propositional probabilistic logic program (L, F) where no two probabilistic facts share the same atom induces a probability mass function p_F over (non-probabilistic) logic programs $L \cup C$, with $C \in \mathcal{T}(F)$, by

$$p_F(C) = \prod_{\mu :: A \in F \mid A \in C} \mu \prod_{\mu :: A \in F \mid A \notin C} (1 - \mu)\,.$$

If L is stratified and normal, then each $L \cup C$ is also stratified and normal, and therefore has at most one answer set [18,20]. Assuming (L, F) is consistent, we can therefore extend P to a probability measure over the algebra of interpretations I of the program by [17]:

$$P(I) = P(\{L \cup C \mid I \in \mathcal{AS}(L \cup C)\}) = \sum_{C \in \mathcal{T}(F) \mid I \in \mathcal{AS}(L \cup C)} p_F(C)\,, \tag{1}$$

where $P(I) = 0$ iff I is not an answer set of any induced program $L \cup C$.

The probability of more complex queries can be computed by defining random variables over interpretations. A useful type of random variable is obtained as indicator functions of ground atoms. Let A be a ground literal in the Herbrand base, then the indicator random variable of A is

$$\mathbb{I}_A(I) = \begin{cases} 1, & \text{if } I \models A; \\ 0, & \text{if } I \not\models A. \end{cases}$$

For convenience, we identify a ground literal with its indicator variable and write simply $P(A)$ to denote $P(\mathbb{I}_A = 1)$. One can check that if A appears solely in a probabilistic fact $\mu :: A$ then $P(A) = \mu$. This notation extends to many random variables; for example $P(A, B)$ denotes the joint probability $P(\mathbb{I}_A = 1, \mathbb{I}_B = 1)$.

Example 7. Consider the probabilistic program in Example 6. Each total choice C induces a logic program that either selects or not a ground atom x, y or z, independently, each with probability 1/2. Hence, $p_F(C) = (1/2)^3$ for every C. Thus, $2^3 \cdot P(\mathsf{phi}) = 3$ counts the number of satisfying assignments of ϕ. It also follows that $P(\mathsf{x}|\mathsf{phi}) = P(\mathsf{phi}, \mathsf{x})/P(\mathsf{phi}) = (1/4)/(3/8) = 2/3$, which agrees with the fact that 2 out of the 3 satisfying assignments of ϕ assign true to X.

3.3. The credal semantics

The semantics of arbitrary consistent probabilistic logic programs is given by probability models (here again we implicitly assume that programs are propositional when defining their semantics, as the semantics of a non-propositional program is the semantics of its grounding). A **probability model** for (L, F) is a probability measure P over the algebra of interpretations such that [4]:

PM1 Every interpretation I with $P(I) > 0$ is an answer set of $L \cup C$, where C is the total choice consistent with I (conversely, I is an extension of C).

PM2 For any total choice C the probability of all the extensions of C to interpretations satisfies

$$P(\{I \mid I \cap C = C\}) = p_F(C) = \prod_{\mu::A \mid A \in C} \mu \prod_{\mu::A \mid A \notin C} (1-\mu)\,.$$

The set of all probability models is called the **credal semantics** of the program [4]. If a program (L, F) is stratified and normal, then the measure in (1) is the unique probability model, and the credal semantics coincides with Sato's distribution semantics [17].

Other semantics for probabilistic answer set programs have been proposed in the literature [14,26,27]. There has also been semantics not based on answer sets; for instance, Hadjichritoudoulou and Warren [5] proposed a semantics based on well-founded models, which define a three-valued semantics over atoms (true, false and undefined). We leave their complexity analysis as future work.

In a previous work [15] we showed that the credal semantics is closed and convex, and corresponds to the set of all probability measures that dominate an infinitely monotone Choquet capacity. As such, the semantics of a ground atom A is characterized by an interval $[\underline{P}(A), \overline{P}(A)]$ such that

$$\underline{P}(A) = \min_P P(A) = \sum_{C \in \mathcal{T}(F) \mid \forall I \in \mathcal{AS}(L \cup C), I \models A} p_F(C)\,, \tag{2}$$

$$\overline{P}(A) = \max_P P(A) = \sum_{C \in \mathcal{T}(F) \mid \exists I \in \mathcal{AS}(L \cup C), I \models A} p_F(C)\,. \tag{3}$$

The right-hand side of Equation (2) collects the probabilities of all induced programs $L \cup C$ for which all answer sets satisfy A. That is, $\underline{P}(A)$ is the sum of the probabilities of the induced programs of which A is a cautious consequence. If A is not a cautious consequence of any induced program $L \cup C$ then $\underline{P}(A) = 0$. Similarly, the right-hand side of Equation (3) collects the probabilities of all induced programs of which A is a brave consequence.

The lower and upper probabilities are tied by the relation $\underline{P}(A) = 1 - \overline{P}(A^c)$, where A^c denotes the complement of A; if A is a literal then A^c denotes its negation.

Example 8. Consider the following probabilistic program:

$$0.5 :: \mathsf{x}. \quad 0.5 :: \mathsf{y}. \tag{ps1}$$

$$\mathsf{z}(0) \vee \mathsf{z}(1). \tag{ps2}$$

$$\mathsf{clause} \leftarrow \mathsf{x}. \quad \mathsf{clause} \leftarrow \mathsf{not\ y}. \tag{ps3}$$

$$\mathsf{phi} \leftarrow \mathsf{clause},\ \mathsf{z}(1). \tag{ps3}$$

First note that $p_F(C) = 1/4$ for any total choice C, and that I is an answer set of $L \cup C$ satisfying phi iff the corresponding assignment to X, Y and Z satisfies the quantified Boolean formula $\phi = (X \vee \neg Y) \wedge Z$. Since for any total choice, there is an answer set that satisfies z(0) but not z(1), we have that $\underline{P}(\mathsf{phi}) = 0$. Also, except for $C = \{\mathsf{y}\}$, all total choices induce programs which satisfy phi, hence $\overline{P}(\mathsf{phi}) = 3 \times (0.5)^2 = 3/4$.

Another property of infinitely monotone Choquet capacities is that the lower and upper conditional probabilities of any event A given B can be written as

$$\underline{P}(A|B) = \min_P P(A|B) = \frac{\underline{P}(A \cap B)}{\underline{P}(A \cap B) + \overline{P}(A^c \cap B)}\,, \tag{4}$$

$$\overline{P}(A|B) = \max_P P(A|B) = \frac{\overline{P}(A \cap B)}{\overline{P}(A \cap B) + \underline{P}(A^c \cap B)}\,, \tag{5}$$

provided that the denominators are positive. For example, for the probabilistic program in Example 8 we have that $\underline{P}(\mathsf{phi}|\mathsf{z}(1)) = 0/(0 + 1/4) = 0$ and $\overline{P}(\mathsf{phi}|\mathsf{z}(1)) = (3/4)/(3/4 + 0) = 1$.

3.4. Languages

As will be shown later, the complexity of probabilistic programs varies with the presence of features in the language. We denote by $\mathsf{Prop}(\mathcal{O})$ the class of propositional programs constructed using language features in $\mathcal{O}$ such as disjunction ($\vee$), negation (not) and aggregate atoms (e.g. sum). We write # to denote that any polynomial-time aggregate function is allowed. We also denote stratified versions by a subscript s; for example $\mathsf{Prop}(\vee, \mathsf{not}_s)$ is the class of propositional aggregate-free negation-stratified disjunctive programs. We denote by $\mathsf{Rel}(\mathcal{O})$ the class of bounded-arity programs using language features in $\mathcal{O}$.

3.5. Inference

We focus on the following computational problems:

Cautious Reasoning (CR): Given a rational number γ, a probabilistic logic program (L, F), and ground literals Q and $E_1, \ldots, E_m$, decide whether $\underline{P}(Q \mid E_1, \ldots, E_m) \geq \gamma$. By convention the answer is negative when $\underline{P}(E_1, \ldots, E_m) = 0$.

This task has received several names in the literature. For example, it has been called simply *inference* [15], MARG [8], and probabilistic query entailment (without evidence) [14]. Some of these works only consider programs with a single answer set for any total choice (as in the case of stratified programs). We felt that cautious reasoning is a more appropriate name, as it reflects its similarity to the analogous reasoning in non-probabilistic answer set programming (which it subsumes), and distinguishes it from a similar task that could be made involving the upper probability (which we could call brave reasoning by analogy). Note that we can decide whether $\overline{P}(Q \mid E_1, \ldots, E_m) \leq \gamma$ by deciding whether $\underline{P}(Q^c \mid E_1, \ldots, E_m) \geq 1 - \gamma$, where Q^c is the complement literal of Q, so that one problem reduces to the other one; we use this fact to avoid analyzing the complexity of computing upper probability bounds.

Most Probable Explanation (MPE): Given a rational number γ, a probabilistic logic program (L, F) and ground literals $E_1, \ldots, E_m$, decide whether $\max_{m_1, \ldots, m_n} \underline{P}(M_1 = m_1, \ldots, M_n = m_n, E_1 = 1, \ldots, E_m = 1) > \gamma$, where $M_1, \ldots, M_n$ are indicator variables for all ground atoms in the Herbrand base of the program.

This is a common task in probabilistic graphical models, that resembles (but it is not equivalent) abduction in logic programming. The idea is to use the maximizing assignments $m_1^*, \ldots, m_n^*$ as a most probable explanation of the observed phenomenon $E_1, \ldots, E_m$ (hence the name).

Maximum a Posteriori Inference (MAP): Given a rational number γ, a probabilistic logic program (L, F), ground literals $Q_1, \ldots, Q_n$ and $E_1, \ldots, E_m$, decide whether $\max_{q_1, \ldots, q_n} \underline{P}(Q_1 = q_1, \ldots, Q_n = q_n, E_1 = 1, \ldots, E_m = 1) > \gamma$.

The motivation for this task is similar to the MPE task, except that here we consider that some atoms are not to be explained; in other words, we want a most probable explanation of $E_1, \ldots, E_n$ marginalizing out atoms that are not in E_i's or in Q_i's.

4. Complexity results

In this section, we present the main contributions of this work: the analysis of the computational complexity of cautious reasoning, most probable explanation and maximum a posteriori inferences parameterized by language features. We assume the reader is familiar with complexity theory [28], in particular with probabilistic Turing machines such as PP [29].

Our results classify languages into complexity classes in Wagner's Counting Hierarchy [30,31]; this is defined as the collection of classes that includes P and such that if C is in the hierarchy then so are the classes of decision problems computed by oracle machines PP^{C}, NP^{C} and $\mathsf{coNP}^{\mathsf{C}}$. The hierarchy therefore contains the Polynomial Hierarchy [32], which includes classes such as $\Sigma_k^{\mathsf{p}} = \mathsf{NP}^{\Sigma_{k-1}^{\mathsf{p}}} = \mathsf{NP}^{\Pi_{k-1}^{\mathsf{p}}}$, $\Pi_k^{\mathsf{p}} = \mathsf{coNP}^{\Sigma_{k-1}^{\mathsf{p}}} = \mathsf{coNP}^{\Pi_{k-1}^{\mathsf{p}}}$ and $\Delta_k^{\mathsf{p}} = \mathsf{P}^{\Sigma_{k-1}^{\mathsf{p}}} = \mathsf{P}^{\Pi_{k-1}^{\mathsf{p}}}$, and also counting classes with oracles in the polynomial hierarchy, such as $\mathsf{PP}^{\Sigma_k^{\mathsf{p}}} = \mathsf{PP}^{\Pi_k^{\mathsf{p}}}$. The latter classes are particularly important for this work.

The complexity results we obtain are summarized in Table 1. We omit results about the complexity of MAP inference, as these are fairly regular: MAP is $\mathsf{NP}^{\mathsf{PP}}$-complete for propositional programs and $\mathsf{NP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete for relational programs. Most results appear for the first time in the literature: the complexity of cautious reasoning for normal programs was established in a previous work [15]; the complexity of cautious reasoning, most probable explanation and maximum a posteriori for normal and disjunctive programs without aggregates appeared in [16]. Among the latter, some results concerning most probable explanation had flaws in their proofs that are corrected in this paper.

From the table, we see that both default negation and negation lead to higher complexity, often requiring extra oracles. The results show that aggregates display the same complexity as disjunction and negation combined; this is a consequence of the fact that one can encode negation and disjunction using aggregate atoms. The complexity of programs with bounded-arity predicates requires an extra NP oracle to "ground" rules; when aggregates are present an additional oracle is also needed to "ground" the atoms inside aggregate atoms. Remarkably, the complexity of the latter grounding depends on the type of aggregate function used: counting and summing demand a PP machine to count over ground atoms, while maximization requires "only" a NP machine. To our knowledge, this property has not been observed before in the literature of

Table 1
The complexity of probabilistic answer set programming under credal semantics. Each line corresponds to a class of programs with indicated constructs: disjunctive heads ($\vee$), aggregates #, and default negation (not). A subindex s denotes stratification w.r.t. that operator. A column with a label containing the subscript all indicate results for bounded-arity relational programs using aggregate symbols sum, count and max, and columns whose label contains subscript max show results for programs using only max. Results with $\geq$ indicate that only hardness has been obtained (memberships for those cases can be taken from that of Rel({not, $\vee$, #})).

	PROPOSITIONAL		BOUNDED-ARITY			
LANGUAGE	CR	MPE	CR_{all}	CR_{max}	MPE_{all}	MPE_{max}
{}	PP	NP	PP^{NP}	PP^{NP}	Σ_2^p	Σ_2^p
$\{\text{not}_s\}$	PP	NP	PP^{NP}	PP^{NP}	Σ_2^p	Σ_2^p
{not}	PP^{NP}	Σ_2^p	$\text{PP}^{\Sigma_2^p}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	Σ_3^p
$\{\vee\}$	PP^{NP}	Σ_3^p	$\text{PP}^{\Sigma_2^p}$	$\text{PP}^{\Sigma_2^p}$	Σ_4^p	Σ_4^p
$\{\text{not}_s, \vee\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^p}$	$\text{PP}^{\Sigma_3^p}$	Σ_4^p	Σ_4^p
$\{\text{not}, \vee\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^p}$	$\text{PP}^{\Sigma_3^p}$	Σ_4^p	Σ_4^p
$\{\#_s\}$	PP	NP	$\text{PP}^{\text{NP}^{\text{PP}}}$	$\text{PP}^{\Sigma_2^p}$	NP^{PP}	Σ_2^p
$\{\text{not}_s, \#_s\}$	PP	NP	$\text{PP}^{\text{NP}^{\text{PP}}}$	$\text{PP}^{\Sigma_2^p}$	NP^{PP}	Σ_2^p
$\{\text{not}, \#_s\}$	PP^{NP}	Σ_2^p	$\text{PP}^{\Sigma_2^{\text{pPP}}}$	$\text{PP}^{\Sigma_3^p}$	Σ_3^{pPP}	Σ_4^p
$\{\vee, \#_s\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\geq \text{PP}^{\Sigma_2^{\text{pPP}}}$	$\geq \text{PP}^{\Sigma_3^p}$	$\geq \Sigma_4^p$	$\geq \Sigma_4^p$
$\{\text{not}_s, \vee, \#_s\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\geq \text{PP}^{\Sigma_2^{\text{pPP}}}$	$\geq \text{PP}^{\Sigma_3^p}$	$\geq \Sigma_4^p$	$\geq \Sigma_4^p$
$\{\text{not}, \vee, \#_s\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\geq \text{PP}^{\Sigma_2^{\text{pPP}}}$	$\geq \text{PP}^{\Sigma_3^p}$	$\geq \Sigma_4^p$	$\geq \Sigma_4^p$
{#}	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^{\text{pPP}}}$	$\text{PP}^{\Sigma_4^p}$	Σ_4^{pPP}	Σ_5^p
$\{\text{not}_s, \#\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^{\text{pPP}}}$	$\text{PP}^{\Sigma_4^p}$	Σ_4^{pPP}	Σ_5^p
{not, #}	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^{\text{pPP}}}$	$\text{PP}^{\Sigma_4^p}$	Σ_4^{pPP}	Σ_5^p
$\{\vee, \#\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^{\text{pPP}}}$	$\text{PP}^{\Sigma_4^p}$	Σ_4^{pPP}	Σ_5^p
$\{\text{not}_s, \vee, \#\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^{\text{pPP}}}$	$\text{PP}^{\Sigma_4^p}$	Σ_4^{pPP}	Σ_5^p
$\{\text{not}, \vee, \#\}$	$\text{PP}^{\Sigma_2^p}$	Σ_3^p	$\text{PP}^{\Sigma_3^{\text{pPP}}}$	$\text{PP}^{\Sigma_4^p}$	Σ_4^{pPP}	Σ_5^p

logic programming. According to these results, while using aggregates often leads to more concise and readable programs, they add significant complexity in the presence of variables, and need to be used with care.

In the rest of this section, we present proofs of the complexity results.

4.1. Cautious reasoning

We start by analyzing the complexity of cautious reasoning. We first establish the upper bound on the complexity (membership), then prove the lower bound (hardness).

4.1.1. Membership

We organize the results from the most general to the more specialized.

The next result establishes membership of probabilistic inference parameterized by the underlying complexity of logical cautious reasoning.

Theorem 1. *Suppose that a class of bounded-arity programs $\mathcal{P}$ is such that logical cautious reasoning without probabilistic facts is in complexity class* C. *Then (probabilistic) cautious reasoning for programs in $\mathcal{P}$ is in* PP^{C}.

Proof. First apply the same polynomial-time reduction as in [15, Theorem 16], to obtain a new evidence-free cautious reasoning problem that is equivalent to the original. Thus, assume that there is no evidence. Fix a total choice C and obtain the nonprobabilistic program $L \cup C$; as the predicates have bounded arity, any total choice is of polynomial size in the size of the program (which includes the number of rules, atoms and constants). Computing the respective lower probability amounts to running (logical) cautious reasoning in this program and collecting the corresponding probability values if the query is a cautious consequence. □

To our knowledge, the complexity of logical reasoning with aggregate atoms and variables has not been established. The next result fills some of the gap. Unlike the propositional case [18], the complexity of reasoning with aggregates depends on the aggregate function used.

Theorem 2. *For programs with no probabilistic facts, cautious reasoning is in*

(a) $\Pi_3^{p\,PP}$ *for programs in* Rel($\vee$, not, #count, #sum, #max)*;*

(b) Π_4^p *for programs in* Rel($\vee$, not, #max)*;*
(c) $\Pi_2^{p\,\mathsf{PP}}$ *for programs in* Rel(not, #sum$_s$)*;*
(d) Π_3^p *for programs in* Rel(not, #max$_s$)*;*
(e) $\Delta_2^{p\,\mathsf{PP}}$ *for programs in* Rel(not$_s$, #sum$_s$)*;*
(f) Δ_3^p *for programs in* Rel(not$_s$, #max$_s$)*;*

Proof. To prove (a) and (b), consider first the problem of deciding whether an interpretation I is *not* an answer set of a program P. This is the case either (i) if I is not a model of the reduct of P w.r.t. I, or (ii) if there is a model $I' \subset I$ of the reduct of P w.r.t. I. To verify (i), guess a rule r and a substitution for the global variables θ, then check if I satisfies all the literals in the body of $r\theta$ and none in the head. We can check if I satisfies an aggregate atom with aggregate symbol sum or count with a P^{PP} machine that adds up the weights of the groundings of symbolic sets that are true w.r.t. I. Likewise, we can check whether I satisfies an aggregate atom with aggregate symbol max with an NP machine. And we can check if I satisfies a standard literal in polynomial time. Hence, (i) can be performed with $\mathsf{NP}^{\mathsf{PP}}$ effort for aggregate symbols sum and count, and with Σ_2^p effort for aggregate symbol max. To verify (ii), guess an interpretation $I' \subset I$ and check whether it is a model of the reduct of P. The latter can be performed by solving (i) and then negating the answer. Hence, the total procedure is accomplished with a $\Sigma_2^{p\,\mathsf{PP}}$ machine for aggregate symbols sum and count, and with a Σ_3^p machine for aggregate symbol max. To verify whether there is some answer set which does *not* contain the query (the complementary problem of cautious reasoning), guess an interpretation not containing the query (using a base NP machine) and check if its an answer set using either a $\Sigma_2^{p\,\mathsf{PP}}$ machine (if there are aggregate symbols count or sum), or a Σ_3^p machine (if there are aggregate symbols max). Negating this decision solves cautious reasoning and thus obtains the desired result.

To show (c) and (d), note that the reduct of a non-disjunctive aggregate-stratified program is equivalent to the (modified) Gelfond-Lifschitz reduct [21], which obtains a positive reduced program P^I by discarding unsatisfied rules, deleting satisfied negative literals and aggregate atoms in the remaining rules, and converting constraints into positive rules (e.g., by inserting a dummy atom in the head which is not satisfied by I). Hence, we can determine if an interpretation I is an answer set of P by verifying (i) if I is a model of its Gelfond-Lifschitz reduct P^I, and if so (ii) if I is minimal in satisfying P^I. As the reduct is definite, we can decide if I is a model of P^I with polynomially many calls to either a $\mathsf{NP}^{\mathsf{PP}}$ oracle (if there are aggregate symbols sum), or to a Σ_2^p oracle (if there are only max symbols). We do so by deriving all true atoms using a stratum to guide the application of rules and the oracle machines to "ground" rules (and checking for violated constraints or contradictions). We can decide if a model I is minimal by finding a founded proof for each atom in I, that is, a sequence of applications of the rules that derives that atom starting from the facts in the program (and does not violate any constraints). As the reduct is definite, these atoms need to be in any model. We provide such a proof as before, by applying rules following a stratum. Both (i) and (ii) can be performed with polynomially many calls to either a $\mathsf{NP}^{\mathsf{PP}}$ machine (for aggregate atoms sum), or to a Σ_2^p machine (for aggregate atoms max). Now to check if there is an answer set that does *not* satisfy the query, guess an interpretation not containing the query (using a base NP machine) and then verify whether it is an answer set (using either an oracle $\mathsf{P}^{\mathsf{NP}^{\mathsf{PP}}}$ or an oracle $\mathsf{P}^{\Sigma_2^{\mathsf{p}}} = \Delta_3^{\mathsf{p}}$). The desired result is obtained as the complement of that decision.

To show (e) and (f), recall that non-disjunctive stratified programs have at most one answer set. As before, we can obtain such an answer set, if it exists, or show that none exists, by using a stratum to guide the application of rules and oracle machines to "ground" the rules: we use an oracle $\mathsf{NP}^{\mathsf{PP}}$ if the rule contains aggregate symbols sum or count, and an oracle Σ_2^p otherwise. At the end we have either a founded proof for each atom in the answer set, or a violated constraint for a derived atom, which shows that no answer set exists. Thus, we solve cautious reasoning with either a polynomial number of calls to $\mathsf{NP}^{\mathsf{PP}}$ or to Σ_2^{p}. □

We can now prove upper bounds on the complexity of bounded-arity and propositional probabilistic logic programs.

Theorem 3. *Cautious reasoning is in*

(a) $\mathsf{PP}^{\Sigma_3^{\mathsf{p}}\,\mathsf{PP}}$ *for programs in* Rel(not, $\vee$, #count, #sum, #max)*;*
(b) $\mathsf{PP}^{\Sigma_4^{\mathsf{p}}}$ *for programs in* Rel(not, $\vee$, #max)*;*
(c) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}\,\mathsf{PP}}$ *for programs in* Rel(not, #sum$_s$)*;*
(d) $\mathsf{PP}^{\Sigma_3^{\mathsf{p}}}$ *for programs in* Rel(not, #max$_s$)*;*
(e) $\mathsf{PP}^{\mathsf{NP}^{\mathsf{PP}}}$ *for programs in* Rel(not$_s$, #sum$_s$)*;*
(f) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$ *for programs in* Rel(not$_s$, #max$_s$)*;*
(g) $\mathsf{PP}^{\Sigma_3^{\mathsf{p}}}$ *for programs in* Rel(not, $\vee$)*;*
(h) $\mathsf{PP}^{\mathsf{NP}}$ *for programs in* Rel(not$_s$)*;*
(i) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$ *for programs in* Prop(not, $\vee$, #)*;*
(j) $\mathsf{PP}^{\mathsf{NP}}$ *for programs in* Prop(not, #$_s$) *or in* Prop($\vee$)*;*
(k) PP *for programs in* Prop(not$_s$, #$_s$).

Proof. All cases follow from Theorem 1 and the respective complexity of logical cautious reasoning: (a)–(f) follows from Theorem 2 (note that $\mathsf{PP}^{\mathsf{PP}^{\Delta_2^{\mathsf{p}}}} = \mathsf{PP}^{\mathsf{NP}^{\mathsf{PP}}}$ and $\mathsf{PP}^{\mathsf{P}^{\Sigma_2^{\mathsf{p}}}} = \mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$); (g) and (h) follows from [25, Table 5] (note that $\mathsf{PP}^{\Delta_2^{\mathsf{p}}} = \mathsf{PP}^{\mathsf{NP}}$); (i)–(k) follows from [18, Table 1] (note that a negative literal can be rewritten as an aggregate atom, and that $\mathsf{PP}^{\mathsf{coNP}} = \mathsf{PP}^{\mathsf{NP}}$). □

Note from the previous result that aggregates introduce the same complexity as disjunction (and interact with negation in similar ways) when the program is propositional; however, for bounded-arity programs, the upper bound varies depending on the type of aggregate symbol used, with max being of "lower complexity" than sum or count.

4.1.2. Hardness

We now prove lower bounds on the complexity of cautious reasoning. Most of the results are obtained by a many-one reduction from quantified Boolean decision problems of the form:

$$Q_1\mathbf{X}_1\, Q_2\mathbf{X}_2 \cdots Q_n\mathbf{X}_n\, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3})\,,$$

where each $\mathbf{X}_i$ denotes a list of variables being quantified over, and Q_i is one of $\exists$, $\forall$ or $\#_{\geq t}$, where the latter denotes "there are at least t assignments", and each L_{ij} is either a variable X_k, its negation $\neg X_k$ or $\bot$. For example, the formula $\#_{\geq 3}X_1, X_2 \exists X_3\, (X_1 \vee X_2) \wedge (\neg X_3 \vee \bot)$ is true, since there are at least 3 assignments to X_1 and X_2 for which there is an assignment to X_3 that satisfies the given formula. Problems of this form are complete for the classes in Wagner's counting hierarchy [30]. For example, the complete problem for NP uses a single quantifier $Q_1 = \exists$, while the complete problem for $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$ uses 3 quantifiers such that $Q_1 = \#_{\geq t}$, $Q_2 = \forall$ and $Q_3 = \exists$.

The proofs are similar to the proofs of hardness for cautious reasoning in non-probabilistic programs [18,33], with the addition of probabilistic facts that "count over" interpretations.

We start with propositional programs.

Theorem 4. *Cautious reasoning is*

(a) PP*-hard for programs in* Prop();
(b) $\mathsf{PP}^{\mathsf{NP}}$*-hard for programs in* Prop($\vee$) *or in* Prop(not);
(c) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$*-hard for programs in* Prop($\mathrm{not}_s, \vee$);
(d) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$*-hard for programs in* Prop($\vee, \#_s$);
(e) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$*-hard for programs in* Prop(#).

Proof. To prove (a) consider a 3-CNF Boolean formula

$$\psi = (L_{11} \wedge L_{12} \wedge L_{13}) \vee \cdots \vee (L_{m1} \wedge L_{m2} \wedge L_{m3})\,,$$

where L_{ij} are literals over variables in $\mathbf{X}_1$. Denote by M the number of satisfying assignments of ψ. Obtain a new formula ψ' by replacing each occurrence of a literal $\neg X_i$ by a fresh variable Y_i, for $X_i \in \mathbf{X}_1$. Goldsmith et al. [34] showed that the formula

$$\psi'' = \psi' \wedge [\wedge_i (X_i \vee Y_i)] \vee [\vee_i (X_i \wedge Y_i)]$$

has exactly $M + 2^{2n} - 3^n$ satisfying assignments. Note that ψ' can be rewritten in 3-CNF by distributing disjunctions over conjunctions in polynomial time. Hence we can decide if ψ has at least t satisfying assignments by deciding whether ψ'' has at least $t + 2^{2n} - 3^n$ satisfying assignments. So consider a Boolean decision problem:

$$\psi = \#_{\geq t}\mathbf{X}_1\, (X_{11} \wedge X_{12} \wedge X_{13}) \vee \cdots \vee (X_{m1} \wedge X_{m2} \wedge X_{m3})\,,$$

where each $X_{ij} \in \mathbf{X}_1$. From the above reasoning and the fact that counting satisfying assignments of arbitrary 3-CNF formulas is PP-complete [30], we have that ψ is PP-complete. Define $\mu_{ij} = \mathsf{x}(k)$ where X_k is the variable appearing in X_{ij}. Set up the program:

$$0.5 :: \mathsf{x}(i). \qquad \text{for each } X_i \in \mathbf{X}_1 \tag{a1}$$

$$\mathsf{c}_i \leftarrow \mu_{i1}. \quad \mathsf{c}_i \leftarrow \mu_{i2}. \quad \mathsf{c}_i \leftarrow \mu_{i3}. \qquad \text{for } i = 1, \ldots, m \tag{a2}$$

$$\mathsf{phi} \leftarrow \mathsf{c}_1, \ldots, \mathsf{c}_m. \tag{a3}$$

Since the program above is positive and constraint-free (and acyclic), each program induced by a total choice has exactly one answer set. It follows that $\underline{P}(\mathsf{phi}) = P(\mathsf{phi}) \geq t/2^{n_1}$ iff ψ is true, where $n_1 = |\mathbf{X}_1|$.

To prove (b) take a quantified Boolean decision problem of the form

$$\phi = \#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \, (L_{11} \wedge L_{12} \wedge L_{13}) \vee \cdots \vee (L_{m1} \wedge L_{m2} \wedge L_{m3}) \,,$$

which is complete for $\mathsf{PP}^{\mathsf{NP}}$ [30]. For $i = 1, \ldots, m$ and $j = 1, 2, 3$, define

$$\mu_{ij} = \begin{cases} \mathsf{x}(k,0) & \text{if } L_{ij} = \neg X_k, \\ \mathsf{x}(k,1) & \text{if } L_{ij} = X_k, \\ \mathsf{true} & \text{otherwise.} \end{cases}$$

Set up the program:

$$\mathsf{true}. \tag{b0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad \text{for each } X_i \in \mathbf{X}_1 \tag{b1}$$

$$\mathsf{x}(i,0) \vee \mathsf{x}(i,1). \qquad \text{for each } X_i \in \mathbf{X}_{1,2} \tag{b2}$$

$$\mathsf{phi} \leftarrow \mu_{i1}, \mu_{i2}, \mu_{i3}. \qquad \text{for } i = 1, \ldots, m \tag{b3}$$

The rules (b3) encode the conjunctions in the DNF formula in ϕ. Fix a total choice, and consider an answer set I of the induced program. Due to subset-minimality, I contains only one of $\mathsf{x}(i,0)$ or $\mathsf{x}(i,1)$ for each i, and it contains the atoms selected by the total choice (which are facts in the induced program). Hence, there is a one-to-one mapping between assignments to the variables in ϕ and answer sets of the program such the DNF formula in ϕ is true in some assignment iff phi is satisfied in the respective answer set. So fix a total choice inducing an assignment to variables in $\mathbf{X}_1$; phi is a cautious consequence of the induced program iff the DNF formula in ϕ is true for all assignments of $\mathbf{X}_2$. And since any such induced program has probability 2^{-n_1}, where $n_1 = |\mathbf{X}_1|$, it follows that $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$ iff ϕ is true. To prove hardness for Prop(not), replace rule (b2) by rules

$$\mathsf{x}(i,0) \leftarrow \mathsf{not}\ \mathsf{x}(i,1). \qquad \text{for each } X_i \in \mathbf{X}_{1,2} \tag{b2'}$$

$$\mathsf{x}(i,1) \leftarrow \mathsf{not}\ \mathsf{x}(i,0). \qquad \text{for each } X_i \in \mathbf{X}_{1,2} \tag{b2"}$$

Again, $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$ iff ϕ is true.

To prove (c), take the $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$-complete problem

$$\phi = \#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3 \, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3}) \,.$$

Note that the same problem can be rewritten as

$$\#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \underbrace{\neg \forall \mathbf{X}_3 \, (\neg L_{11} \wedge \neg L_{12} \wedge \neg L_{13}) \vee \cdots \vee (\neg L_{m1} \wedge \neg L_{m2} \wedge \neg L_{m3})}_{=\psi} \,.$$

Define μ_{ij} as before, replacing $\neg L_{ij}$ with L_{ij} in the definition (e.g. $\mu_{ij} = \mathsf{x}(k,0)$ if $\neg L_{ij} = \neg X_k$). Set up the program:

$$\mathsf{true}. \tag{c0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad \text{for all } X_i \in \mathbf{X}_1 \tag{c1}$$

$$\mathsf{x}(i,0) \vee \mathsf{x}(i,1). \qquad \text{for all } X_i \in \mathbf{X}_{1,2,3} \tag{c2}$$

$$\mathsf{x}(i,0) \leftarrow \mathsf{dnf}. \quad \mathsf{x}(i,1) \leftarrow \mathsf{dnf}. \qquad \text{for all } X_i \in \mathbf{X}_3 \tag{c3}$$

$$\mathsf{dnf} \leftarrow \mu_{i1}, \mu_{i2}, \mu_{i3}. \qquad \text{for } i = 1, \ldots, m \tag{c4}$$

$$\mathsf{phi} \leftarrow \mathsf{not}\ \mathsf{dnf}. \tag{c5}$$

The rules in (d3) encode the universal quantifier in ψ. To see why this is true, consider an answer set I containing dnf. Due to subset-minimality, I contains only one of $\mathsf{x}(i,0)$ and $\mathsf{x}(i,1)$ for $X_i \in \mathbf{X}_1 \cup \mathbf{X}_2$. And since $\mathsf{dnf} \in I$ then I contains all $\mathsf{x}(i,0)$ and $\mathsf{x}(i,1)$ for $X_i \in \mathbf{X}_3$. Now, as before, each I can be associated with a single assignment $\mathbf{x}_1, \mathbf{x}_2$ to variables in $\mathbf{X}_1$ and $\mathbf{X}_2$. Consider some assignment $\mathbf{x}_3$ to variables $\mathbf{X}_3$. If $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ does not satisfy any of the terms in ψ, then there the interpretation I' that encodes such assignment is a model of the program and a subset of I. As this contradicts the assumption that I is an answer set, we conclude that ψ must be true for such $\mathbf{x}_1$ and $\mathbf{x}_2$ whenever dnf is true in the corresponding answer set. The rest of the program repeats the reasoning in (a). We thus have that $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$ iff ϕ is true.

To prove (d), replace rule (d5) with the rule

$$\mathsf{phi} \leftarrow \#\mathsf{count}\{1 : \mathsf{dnf}\} < 1. \tag{d1}$$

Note that the body of the rule above is true iff dnf is false. Thus, $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$ iff ϕ is true.

To prove (e) take again a formula ϕ and set up the program:

$$\mathsf{true}. \tag{e0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad \text{for } X_i \in \mathbf{X}_1 \tag{e1}$$

$$\mathsf{x}(i,0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} \leq 0. \qquad \text{for } X_i \in \mathbf{X}_{1,2,3} \tag{e2}$$

$$\mathsf{x}(i,1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} \geq 0. \qquad \text{for } X_i \in \mathbf{X}_{1,2,3} \tag{e3}$$

$$\mathsf{x}(i,0) \leftarrow \mathsf{dnf}. \quad \mathsf{x}(i,1) \leftarrow \mathsf{dnf}. \qquad \text{for } X_i \in \mathbf{X}_3 \tag{e4}$$

$$\mathsf{dnf} \leftarrow \mu_{i1}, \mu_{i2}, \mu_{i3}. \qquad \text{for } i = 1, \ldots, m \tag{e5}$$

$$\mathsf{phi} \leftarrow \#\mathsf{sum}\{1 : \mathsf{dnf}\} \leq 0. \tag{e6}$$

The literals μ_{ij} are defined as in (c). The rules (e2) and (e3) are satisfied only if at least one of $\mathsf{x}(i,0)$ or $\mathsf{x}(i,1)$ is true. By minimality, exactly one of these atoms will be true for $X_i \in \mathbf{X}_1 \cup \mathbf{X}_2$. Moreover, both $\mathsf{x}(i,0)$ and $\mathsf{x}(i,1)$ are true iff ψ is true for a given assignment of $\mathsf{x}_1, \mathsf{x}_2$ (and its corresponding interpretation). Once more, we have that $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$ iff ϕ is true. □

We now consider relational programs. Here the complexity varies by the aggregate function used.

Theorem 5. *Cautious reasoning is*

(a) $\mathsf{PP}^{\mathsf{NP}}$*-hard for programs in* $\mathsf{Rel}()$*;*
(b) $\mathsf{PP}^{\Sigma_2^{\mathsf{p}}}$*-hard for programs in* $\mathsf{Rel}(\vee)$*;*
(c) $\mathsf{PP}^{\Sigma_3^{\mathsf{p}}}$*-hard for programs in* $\mathsf{Rel}(\mathsf{not}_s, \vee)$ *or in* $\mathsf{Rel}(\vee, \#\mathsf{max}_s)$*;*
(d) $\mathsf{PP}^{\Sigma_2^{\mathsf{pPP}}}$ *-hard for programs in* $\mathsf{Rel}(\mathsf{not}, \#\mathsf{sum}_s)$, $\mathsf{Rel}(\mathsf{not}, \#\mathsf{count}_s)$, $\mathsf{Rel}(\vee, \#\mathsf{sum}_s)$ *or in* $\mathsf{Rel}(\vee, \#\mathsf{count}_s)$*;*
(e) $\mathsf{PP}^{\Sigma_3^{\mathsf{pPP}}}$ *-hard for programs in* $\mathsf{Rel}(\#\mathsf{sum})$*;*
(f) $\mathsf{PP}^{\Sigma_4^{\mathsf{p}}}$*-hard for programs in* $\mathsf{Rel}(\#\mathsf{max})$.

Proof. We prove (a) by reduction from the $\mathsf{PP}^{\mathsf{NP}}$-complete problem:

$$\phi = \#_{\geq t}\mathbf{X}_1 \exists \mathbf{X}_2 \, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3}) \, .$$

The reduction encodes the existential quantification over $\mathbf{X}_2$ using relational rules, and atoms $\mathsf{x}(i,0)$ and $\mathsf{x}(i,1)$ to "count over" variables in $\mathbf{X}_1$. Thus define

$$\mu_{ij}(X) = \begin{cases} \mathsf{x}(k,0) & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_1; \\ \mathsf{x}(k,1) & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_1; \\ \mathsf{false} & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 1; \\ \mathsf{true} & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 0; \\ \mathsf{true} & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 1; \\ \mathsf{false} & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 0. \end{cases}$$

For example, if the first clause is $\neg X_1 \vee X_2 \vee \neg X_3$, with $X_1 \in \mathbf{X}_1$ and $X_2, X_3 \in \mathbf{X}_2$, then $\mu_{11}(0) = \mu_{11}(1) = \mathsf{x}(1,0)$, $\mu_{12}(0) = \mathsf{false}$, $\mu_{12}(1) = \mathsf{true}$, $\mu_{13}(0) = \mathsf{true}$ and $\mu_{13}(1) = \mathsf{false}$. Now set up the program:

$$\mathsf{true}. \tag{a0}$$

$$0.5 :: \mathsf{x}(i,0). \quad 0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{a1}$$

$$\mathsf{false} \leftarrow \mathsf{x}(i,0), \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{a2}$$

$$\mathsf{c}_i(X_1, X_2, X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1, \ldots, m; \quad j = 1, 2, 3; \; X_j = 0, 1 \tag{a3}$$

$$\mathsf{phi} \leftarrow \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3}). \tag{a4}$$

The rules (a2) identify interpretations that map into inconsistent assignments to $\mathbf{X}_1$. The variable X_j in rule (a3) represents the variable in literal L_{ij} in ϕ. The program is similar to the program in the proof of Theorem 4(a), except that the quantification over variables $\mathbf{X}_2$ is encoded using 3-arity predicates c_i. The rules (a3) encode the clauses in ϕ for each assignment to variables in $\mathbf{X}_2$. So for example, if the first clause is $\neg X_1 \vee X_2 \vee \neg X_3$ as before, then the program contains the rules

$\mathsf{c}_1(0,Y,Z) \leftarrow \mathsf{x}(1,0).$ $\quad \mathsf{c}_1(X,0,Z) \leftarrow \mathsf{false}.$ $\quad \mathsf{c}_1(X,Y,0) \leftarrow \mathsf{true}.$

$\mathsf{c}_1(1,Y,Z) \leftarrow \mathsf{x}(1,0).$ $\quad \mathsf{c}_1(X,1,Z) \leftarrow \mathsf{true}.$ $\quad \mathsf{c}_1(X,Y,1) \leftarrow \mathsf{false}.$

Consider an answer set I for the rules above. If the assignment to X_1 corresponding to $\mathsf{x}(i,v)$, with $i=1,2$ and $v=0,1$, satisfies $\neg X_1 \vee X_2 \vee \neg X_3$, then all groundings of $\mathsf{c}_1(X_1,X_2,X_3)$ are true. On the other hand, if the corresponding assignment to X_1 does not satisfy the clause, then only the groundings of $\mathsf{c}_1(X_1,X_2,X_3)$ with $X_2=1$ or $X_3=0$ are true. This way, the rules above encode satisfying assignments of the respective clause. This reasoning extends to the whole program. Note that we can actually omit rules (a3) when X_{ij} is not in $\mathbf{X}_2$ without altering the semantics of the program. The program above is positive and constraint-free (and acyclic), thus admits exactly one probability model. Let E be an indicator variable on false. We have that $P(\mathsf{false}=0) = 2^{-2n_1}\sum_{k=1}^{n_1}\binom{n_1}{k} = 1-2^{-2n_1}$, where $n_1 = |\mathbf{X}_1|$. It follows that $P(\mathsf{phi}|\mathsf{false}=0) \geq t/(2^{2n_1}-1)$ iff ϕ is true.

We prove (b) by reduction from the $\mathsf{PP}^{\Sigma_2^{\mathrm{p}}}$-complete problem:

$$\phi = \#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3 \, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3}) \, .$$

The reduction is similar to the reduction in (a), with $\mathbf{X}_3$ now playing the role of $\mathbf{X}_2$, except that we now use a disjunction to encode the quantification over the variables in $\mathbf{X}_2$. Define $\mu_{ij}(X)$ as before, replacing $\mathbf{X}_2$ with X_3 and $\mathbf{X}_1$ instead of $\mathbf{X}_{1,2}$. Then set up the program:

$$\mathsf{true}. \tag{b0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{b1}$$

$$\mathsf{x}(i,0) \vee \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_{1,2} \tag{b2}$$

$$\mathsf{c}_i(X_1,X_2,X_3) \leftarrow \mu_{ij}(X_j). \qquad i=1,\ldots,m;\ j=1,2,3;\ X_j=0,1 \tag{b3}$$

$$\mathsf{phi} \leftarrow \mathsf{c}_1(X_{11},X_{12},X_{13}),\ldots,\mathsf{c}_m(X_{m1},X_{m2},X_{m3}). \tag{b4}$$

Cautious reasoning with the above program collects probabilities of the induced programs of which phi is a (logical) cautious consequence. Thus, it follows that ϕ is true iff $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$, where $n_1 = |\mathbf{X}_1|$.

To prove (c), consider the $\mathsf{PP}^{\Sigma_3^{\mathrm{p}}}$-complete problem [30]:

$$\#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \neg \forall \mathbf{X}_3 \exists \mathbf{X}_4 \, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3}) \, .$$

Define $\mu_{ij}(X)$ as in (b), replacing $\mathbf{X}_3$ with $\mathbf{X}_4$. Set up the program:

$$\mathsf{true}. \tag{c0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{c1}$$

$$\mathsf{x}(i,0) \vee \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_{1,2,3} \tag{c2}$$

$$\mathsf{x}(i,0) \leftarrow \mathsf{cnf}. \qquad X_i \in \mathbf{X}_3 \tag{c3}$$

$$\mathsf{x}(i,1) \leftarrow \mathsf{cnf}. \qquad X_i \in \mathbf{X}_3 \tag{c4}$$

$$\mathsf{c}_i(X_1,X_2,X_3) \leftarrow \mu_{ij}(X_j). \qquad i=1,\ldots,m;\ j=1,2,3;\ X_j=0,1 \tag{c5}$$

$$\mathsf{cnf} \leftarrow \mathsf{c}_1(X_{11},X_{12},X_{13}),\ldots,\mathsf{c}_m(X_{m1},X_{m2},X_{m3}). \tag{c6}$$

$$\mathsf{phi} \leftarrow \mathsf{not\ cnf}. \tag{c7}$$

This program combines the ideas of the program in (b) with the ideas of the program in the proof of Theorem 4(c). Therefore, $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$ iff ϕ is true. To prove hardness for $\mathsf{Rel}(\vee, \#\mathsf{max}_s)$ replace rule (d7) by rule $\mathsf{phi} \leftarrow \#\mathsf{max}\{0:\mathsf{true};1:\mathsf{cnf}\} < 1$, and repeat the query.

We prove (d) by reduction from the $\mathsf{PP}^{\Sigma_2^{\mathrm{pPP}}}$-complete problem [30]:

$$\phi = \#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3 \#_{\geq s}\mathbf{X}_4 \, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3}) \, .$$

Define $\mu_{ij}(X)$ as in item (b) with $\mathbf{X}_{3,4}$ instead of X_3, and set up the program:

$$\mathsf{true}.\quad \mathsf{e}(0).\quad \mathsf{e}(1). \tag{d0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{d1}$$

$$\mathsf{x}(i,0) \leftarrow \mathsf{not}\, \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_{1,2} \tag{d2}$$

$$\mathsf{c}_i(X_1,X_2,X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1,\ldots,m \quad j = 1,2,3;\ X_j = 0,1 \tag{d3}$$

$$\mathsf{phi} \leftarrow \mathsf{e}(X_1),\ldots,\mathsf{e}(X_{n_3}), \quad \#\mathsf{sum}\{1, \mathbf{X}_4 : \mathsf{c}_1(X_{11},X_{12},X_{13}),\ldots,\mathsf{c}_m(X_{m1},X_{m2},X_{m3})\} \geq s. \tag{d4}$$

The expression $\mathsf{e}(X_1),\ldots,\mathsf{e}(X_{n_3})$ encodes the existential quantification over variables in $\mathbf{X}_3$; hence rule (d4) encodes $\exists \mathbf{X}_3 \#_{\geq s} \mathbf{X}_4\, \varphi$, where φ denotes the 3-CNF formula in ϕ. We have that ϕ is true iff $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$. Replacing sum with count in (d4) proves the result for $\mathsf{Rel}(\mathsf{not}, \#\mathsf{count}_s)$. To prove hardness for $\mathsf{Rel}(\vee, \#\mathsf{sum}_s)$ replace (d2) by

$$\mathsf{x}(i,0) \leftarrow \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_{1,2} \tag{d2'}$$

Further replacing sum with count in (d4) proves hardness of $\mathsf{Rel}(\vee, \#\mathsf{count}_s)$.

We prove (e) by reduction from the $\mathsf{PP}^{\mathsf{PP}}_2$-complete problem [30]:

$$\phi = \#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3 \forall \mathbf{X}_4 \#_{\geq s} \mathbf{X}_5\, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3})\,.$$

This problem is equivalent to

$$\#_{\geq t}\mathbf{X}_1 \forall \mathbf{X}_2 \neg \forall \mathbf{X}_3 \exists \mathbf{X}_4 \#_{< s} \mathbf{X}_5\, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee L_{m2} \vee L_{m3})\,.$$

Define:

$$\mu_{ij}(X) = \begin{cases} \mathsf{x}(k,0) & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_{1,2,3}; \\ \mathsf{x}(k,1) & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_{1,2,3}; \\ \mathsf{false} & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_{4,5} \text{ and } X = 1; \\ \mathsf{true} & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_{4,5} \text{ and } X = 0; \\ \mathsf{true} & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_{4,5} \text{ and } X = 1; \\ \mathsf{false} & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_{4,5} \text{ and } X = 0. \end{cases}$$

Then assemble the program:

$$\mathsf{true}.\quad \mathsf{e}(0).\quad \mathsf{e}(1). \tag{e0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{e1}$$

$$\mathsf{x}(i,0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} \leq 0. \qquad X_i \in \mathbf{X}_{1,2,3,4} \tag{e2}$$

$$\mathsf{x}(i,1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} \geq 0. \qquad X_i \in \mathbf{X}_{1,2,3,4} \tag{e3}$$

$$\mathsf{x}(i,0) \leftarrow \mathsf{phi}. \qquad X_i \in \mathbf{X}_3 \tag{e3}$$

$$\mathsf{x}(i,1) \leftarrow \mathsf{phi}. \qquad X_i \in \mathbf{X}_3 \tag{e4}$$

$$\mathsf{c}_i(X_1,X_2,X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1,\ldots,m \quad j = 1,2,3;\ X_j = 0,1 \tag{e5}$$

$$\mathsf{psi} \leftarrow \mathsf{e}(X_1),\ldots,\mathsf{e}(X_{n_4}), \quad \#\mathsf{sum}\{1, \mathbf{X}_5 : \mathsf{c}_1(X_{11},X_{12},X_{13}),\ldots,\mathsf{c}_m(X_{m1},X_{m2},X_{m3})\} < s. \tag{e6}$$

$$\mathsf{phi} \leftarrow \#\mathsf{sum}\{1 : \mathsf{psi}\} < 1. \tag{e7}$$

The variables $X_1,\ldots,X_{n_4}$ correspond to the variables in $\mathbf{X}_4$ (the atoms e encode the existential quantification over these variables). Rule (e7) encodes $\psi = \neg\forall \mathbf{X}_4 \#_{\geq s} \varphi$, where φ denotes the 3-CNF formula in ϕ. The aggregate atom is used to encode negation (i.e., phi is true iff psi is false). We have that ψ is true iff $\underline{P}(\mathsf{phi}) \geq t/2^{n_1}$.

To prove (f), use the program formed by rules (e0)–(e1), (e3)–(e5), constraints

$$\leftarrow \#\mathsf{max}\{1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} < 1. \qquad X_i \in \mathbf{X}_{1,2,3,4} \tag{f2}$$

and rules

$$\mathsf{psi} \leftarrow \mathsf{e}(X_1), \ldots, \mathsf{e}(X_{n_4}), \tag{f6}$$
$$\#\max\{1, \mathbf{X}_5 : \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3})\} < 1.$$
$$\mathsf{phi} \leftarrow \#\max\{0 : \mathsf{true}; 1 : \mathsf{psi}\} < 1. \tag{f7}$$

The constraints in (f2) are satisfied iff a interpretation satisfies rules (e2). □

4.2. Most probable explanation

We now move to the complexity of MPE. As before, we first establish upper-bounds on the complexity, then prove the lower-bounds.

4.2.1. Membership

To get some insight into the problem, consider a propositional program (L, F) and a total choice C, inducing a non-probabilistic program $L \cup C$. If there is a single answer set I for such program, then by Properties PM1 and PM2 we have that $\underline{P}(I) = \prod_{A \in C} \mu_A \prod_{A \notin C} (1 - \mu_A)$. If there is more than one answer set, then for each answer set $I \in \mathcal{AS}(L \cup C)$ there is a probability model P assigning $P(I) = 0$, whence $\underline{P}(I) = 0$. So MPE can be decided by going through each total choice, verifying whether the induced logical program has a single answer set I consistent with the evidence. If it has, then we check whether the probability of the respective total choice $C \subseteq I$ is greater than the given threshold. If no total choice induces a unique answer set consistent with the evidence, then the lower probability of any interpretation is zero. This leads us to the following result.

Theorem 6. *MPE inference is in:*

(a) $\Sigma_4^{p\,\mathsf{PP}}$ *for programs in* Rel(not, $\vee$, #count, #sum, #max)*;*
(b) Σ_5^p *for programs in* Rel(not, $\vee$, #max)*;*
(c) $\Sigma_3^{p\,\mathsf{PP}}$ *for programs in* Rel(not, $\#\mathsf{sum}_s$)*;*
(d) Σ_4^p *for programs in* Rel(not, $\#\mathsf{max}_s$)*;*
(e) Σ_4^p *for programs in* Rel(not, $\vee$)*;*
(f) $\mathsf{NP}^{\mathsf{PP}}$ *for programs in* Rel(not_s, $\#\mathsf{sum}_s$)*;*
(g) Σ_2^p *for programs in* Rel(not_s)*;*
(h) Σ_3^p *for programs in* Prop(not, $\vee$, #)*;*
(i) Σ_2^p *for programs in* Prop(not, $\#_s$)*; and*
(j) NP *for programs in* Prop(not_s, $\#_s$).

Proof. Note that all results consist of a base NP machine with different oracles, and that we have assumed that probabilistic programs are always consistent. So guess an interpretation I and reject if it does not satisfy the evidence (this takes polynomial time). Let C be the total choice consistent with I. There are two scenarios to consider: If there is a single answer set $I' \subseteq I$ consistent with C, then $\underline{P}(I') = p_F(C)$. Otherwise, we have that $\underline{P}(I) = 0$.

So to decide whether $\underline{P}(I) > \gamma$ we need to verify whether there is an answer set consistent with C and different than I. This can be accomplished by extending the logical program L with a fact A for each $A \in C$, a constraint $\leftarrow A$ for $A \notin C$, and a constraint $\leftarrow L_1, \ldots, L_m$, where each L_i represents the assignment to an atom in I: $L_i = \mathsf{not}\, A$ if $I \not\models A$, and $L_i = A$ if $I \models A$. The extended program admits an answer set iff there exists an answer set consistent with C and not equal to I. Hence, we can compute MPE inference with an NP base machine that goes through each interpretation, equipped with an oracle that solves answer set existence for the corresponding non-probabilistic language. As the latter problem can be reduced to cautious reasoning (simply add a fresh atom and query if it is a cautious consequence), we can use the results from Theorem 2, [25] and [18] to show membership of all cases. □

Membership for aggregate-free programs was proved with a minor mistake in [16]: cautious reasoning was used to verify if an interpretation is an answer set instead of verifying if it is the unique answer set. The proof above rectifies that result.

4.2.2. Hardness

We now prove the lower-bound on the complexity of MPE. We start with propositional programs:

Theorem 7. *MPE inference is*

(a) NP*-hard for programs in* Prop()*;*
(b) Σ_2^p*-hard for programs in* Prop(not)*; and*
(c) Σ_3^p*-hard for programs in* Prop($\vee$) *or in* Prop(#).

Proof. To show (a), consider the NP-complete problem of deciding if there exists an *independent set* of a graph $G = (G_V, G_E)$ of size k [28].[2] To solve that problem, assemble the program:

$$3/4 :: \mathsf{vertex}(i). \qquad i \in G_V \tag{a1}$$

$$\mathsf{fault} \leftarrow \mathsf{vertex}(i), \mathsf{vertex}(j). \qquad (i,j) \in G_E \tag{a2}$$

Each total choice corresponds to a candidate independent set. Rules (a2) encode the constraints that no two vertices in an independent set can share an edge in G_E. Let $M_1, \ldots, M_n$ be indicator variables for $\mathsf{vertex}(i)$, $i \in G_V$, and E be an indicator variable for fault. Then $\max_{m_1,\ldots,m_n} \underline{P}(M_1 = m_1, \ldots, M_n = m_n, E = 0) > 3^{k-1}/4^n$ iff G has an independent set of size k.

To prove (b), consider the Σ_2^p-complete problem $\exists \mathbf{X}_1 \forall \mathbf{X}_2 \phi$, where ϕ is in 3-DNF with m terms $L_{i1} \wedge L_{i2} \wedge L_{i3}$. Let $X_1, \ldots, X_n$ be the variables in $\mathbf{X}_2$, and X_{n+1} be a fresh variable not in $\mathbf{X}_1$ or $\mathbf{X}_2$. For each assignment to $\mathbf{X}_1$, the quantified Boolean formula $\forall \mathbf{X}_2 \phi$ is true iff there is a unique assignment to $\mathbf{X}_2$ and X_{n+1} that satisfies

$$\psi = (\neg X_{n+1} \wedge \neg\phi) \vee (X_1 \wedge \cdots \wedge X_n \wedge X_{n+1}).$$

Note that $\neg\phi$ is in 3-CNF with clauses $\neg L_{i1} \vee \neg L_{i2} \vee \neg L_{i3}$, $i = 1, \ldots, m$. Set up the program:

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{b1}$$

$$\mathsf{x}(i,0) \leftarrow \mathsf{not}\ \mathsf{x}(i,1). \qquad \mathsf{x}(i,1) \leftarrow \mathsf{not}\ \mathsf{x}(i,0). \qquad X_i \in \mathbf{X}_{1,2} \tag{b2}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(n+1,0), \mathsf{nphi}. \tag{b3}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(0,1), \ldots, \mathsf{x}(n+1,1). \tag{b4}$$

$$\mathsf{nphi} \leftarrow \neg L_{i1}. \quad \mathsf{nphi} \leftarrow \neg L_{i2}. \quad \mathsf{nphi} \leftarrow \neg L_{i3}. \qquad i = 1, \ldots, m \tag{b5}$$

The rules (c5) encode the clauses in $\neg\phi$ using atoms $\mathsf{x}(i,0)$ and $\mathsf{x}(i,1)$. So suppose I is an answer set that satisfies psi and does *not* satisfy (b4) (hence I satisfies nphi and $\mathsf{x}(n+1,0)$). By (b2), I contains at most one of $\mathsf{x}(i,0)$ or $\mathsf{x}(i,1)$ and thus encodes an assignment to variables X_i. By minimality of I, such an assignment must satisfy $\neg\phi$. Now fix a total choice C. The corresponding program has exactly one answer set satisfying psi iff there is an assignment to $\mathbf{X}_1$ induced by C for which only the assignment that assigns true to all of $X_1, \ldots, X_n$ satisfies ψ. Thus fix evidence E denoting the atom psi, and let $M_1, \ldots, M_n$ be indicator variables representing the atoms in the program. There is a configuration $m_1, \ldots, m_n$ such that $\underline{P}(m_1, \ldots, m_n, E = 1) > 0$ iff there is an assignment to $\mathbf{X}_1$ such that there is exactly one assignment to $X_1, \ldots, X_{n+1}$ satisfying ψ.

To prove (c), take a formula $\alpha = \exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3 \phi$, where ϕ is in 3-CNF with m clauses. Let $X_1, \ldots, X_n$ be the variables in $\mathbf{X}_2$, and let X_{n+1} be a fresh variable not in $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$. Then α is true iff there is an assignment to $\mathbf{X}_1$ such that there is a unique assignment to $\mathbf{X}_2$ that satisfies

$$\psi = (\neg X_{n+1} \wedge \forall \mathbf{X}_3 \neg\phi) \vee (X_1 \wedge \cdots \wedge X_{n+1}).$$

Set up the program formed:

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{c1}$$

$$\mathsf{x}(i,0) \vee \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_{1,2,3} \tag{c2}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(n+1,0), \mathsf{nphi}. \tag{c3}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(0,1), \ldots, \mathsf{x}(n+1,1). \tag{c4}$$

$$\mathsf{nphi} \leftarrow L_{i1}, L_{i2}, L_{i3}. \qquad i = 1, \ldots, m \tag{c5}$$

$$x(i,0) \leftarrow \mathsf{nphi}. \qquad \mathsf{x}(i,1) \leftarrow \mathsf{nphi}. \qquad X_i \in \mathbf{X}_3 \tag{c6}$$

Fix a total choice C and a corresponding assignment to $\mathbf{X}_1$. The program $L \cup C$ has exactly one answer set iff there is a unique assignment to $\mathbf{X}_2$ (assigning true to $X_1, \ldots, X_n$) that satisfies $\forall \mathbf{X}_2 \neg\phi$. We have that α is true iff $\max_{m_1,\ldots,m_n} \underline{P}(M_1 = m_1, \ldots, M_n = m_n, \mathsf{psi}) > 0$, where $M_1, \ldots, M_n$ are indicator variables on all atoms in the program.

Finally, to prove hardness for Prop(#), replace (d2) with

$$\mathsf{x}(i,0) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} \leq 0. \qquad X_i \in \mathbf{X}_{1,2,3} \tag{c2'}$$

$$\mathsf{x}(i,1) \leftarrow \#\mathsf{sum}\{-1 : \mathsf{x}(i,0); 1 : \mathsf{x}(i,1)\} \geq 0. \qquad X_i \in \mathbf{X}_{1,2,3} \tag{c2''}$$

The MPE inference is positive iff the original problems is satisfiable. □

[2] An independent set is a subset of nonadjacent vertices of the graph.

In a previous work [16], we showed a proof of Σ_2^p-hardness of normal programs without aggregates by a reduction from the quantified Boolean satisfiability problem $\exists \mathbf{X}_1 \forall \mathbf{X}_2 \, \phi$, where ϕ is a quantifier-free Boolean formula. That proof is however incorrect when the formula ϕ has more than a single model sharing the assignment to the variables in $\mathbf{X}_1$. The previous result corrects that proof.

We now establish hardness for programs with bounded-arity predicates. All results follow by a combination of the ideas used to prove Theorems 5 and 7.

Theorem 8. *MPE inference is*

(a) Σ_2^p*-hard for programs in* Rel();
(b) $\mathsf{NP}^{\mathsf{PP}}$*-hard for programs in* Rel(#sum$_s$);
(c) Σ_3^p*-hard for programs in* Rel(not);
(d) Σ_4^p*-hard for programs in* Rel($\vee$);
(e) $\Sigma_3^{p\mathsf{PP}}$*-hard for programs in* Rel(not, #sum$_s$);
(f) Σ_4^p*-hard for programs in* Rel(not, #max$_s$);
(g) $\Sigma_4^{p\mathsf{PP}}$*-hard for programs in* Rel(#sum);
(h) Σ_5^p*-hard for programs in* Rel($\vee$, #max).

Proof. To show (a), consider the Σ_2^p-complete problem:

$$\phi = \exists \mathbf{X}_1 \forall \mathbf{X}_2 \, (\neg L_{11} \wedge \neg L_{12} \wedge \neg L_{13}) \vee \cdots \vee (\neg L_{m1} \wedge \cdots \neg L_{m3}) \,,$$

where each L_{ij} is a literal over the variables in $\mathbf{X}_{1,2}$. This problem is equivalent to

$$\phi = \exists \mathbf{X}_1 \neg \exists \mathbf{X}_2 \, (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee \cdots L_{m3}) \,,$$

where the Boolean expression is in 3-CNF. Define:

$$\mu_{ij}(X) = \begin{cases} \mathsf{x}(k,0) & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_1, \\ \mathsf{x}(k,1) & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_1, \\ \mathsf{false} & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 1, \\ \mathsf{true} & \text{if } L_{ij} = \neg X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 0, \\ \mathsf{true} & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 1, \\ \mathsf{false} & \text{if } L_{ij} = X_k \text{ and } X_k \in \mathbf{X}_2 \text{ and } X = 0. \end{cases}$$

Set up the program:

$$\mathsf{true}. \tag{a0}$$

$$0.5 :: \mathsf{x}(i,0). \quad 0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{a1}$$

$$\mathsf{false} \leftarrow \mathsf{x}(i,0), \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{a2}$$

$$\mathsf{c}_i(X_1, X_2, X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1, \ldots, m \tag{a3}$$
$$j = 1, 2, 3; \; X_j = 0, 1$$

$$\mathsf{cnf} \leftarrow \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3}). \tag{a4}$$

The program above is positive and consistent, hence admits a single probability model. It follows that the MPE inference with evidence $\mathsf{cnf} = 1$ and $\mathsf{false} = 0$ is positive iff ϕ is true.

To prove (b), consider the $\mathsf{NP}^{\mathsf{PP}}$-complete problem

$$\phi = \exists \mathbf{X}_1 \#_{\geq t} \mathbf{X}_2 (L_{11} \vee L_{12} \vee L_{13}) \wedge \cdots \wedge (L_{m1} \vee \cdots L_{m3}) \,,$$

where each L_{ij} is literal over a variable in $\mathbf{X}_1$ or $\mathbf{X}_2$. Define μ_{ij} as in (a) and set up the program:

$$\mathsf{true}. \tag{b0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_1 \tag{b1}$$

$$\mathsf{x}(i,0) \leftarrow \#\mathsf{sum}\{1 : \mathsf{x}(i,1)\} \leq 0. \qquad X_i \in \mathbf{X}_1 \tag{b2}$$

$$\mathsf{c}_i(X_1, X_2, X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1, \ldots, m \tag{b3}$$
$$j = 1, 2, 3; \; X_j = 0, 1$$

$$\mathsf{cnf} \leftarrow \#\mathsf{sum}\{1, \mathbf{X}_2 : \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3})\} \geq t. \tag{b4}$$

As the program above is stratified and consistent, the program admits a single probability model. Thus ϕ is true iff the MPE inference with evidence cnf is positive.

To prove (c), consider the Σ_3^p-complete problem $\alpha = \exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3\, \phi$, where ϕ is in 3-CNF with clauses $L_{i1} \vee L_{i2} \vee L_{i3}$, for $i = 1, \ldots, m$. This formula is equivalent to

$$\exists \mathbf{X}_1 \exists! \mathbf{X}_2 X_{n+1} (\neg X_{n+1} \wedge \neg \exists \mathbf{X}_3\, \phi) \vee (X_1 \wedge \cdots \wedge X_{n+1}),$$

where $\exists!$ denotes "there is a single instantiation", $\mathbf{X}_2 = \{X_1, \ldots, X_n\}$, and X_{n+1} is a fresh variable. Set up the program:

$$\mathsf{true}. \tag{c0}$$

$$0.5 :: \mathsf{x}(i, 1). \qquad X_i \in \mathbf{X}_1 \tag{c1}$$

$$\mathsf{x}(i, 0) \leftarrow \mathsf{not}\ \mathsf{x}(i, 1). \qquad X_i \in \mathbf{X}_{1,2,3} \tag{c2}$$

$$\mathsf{x}(i, 1) \leftarrow \mathsf{not}\ \mathsf{x}(i, 0). \qquad X_i \in \mathbf{X}_{1,2,3} \tag{c3}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(n+1, 0), \mathsf{not\ nphi}. \tag{c4}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(0, 1), \ldots, \mathsf{x}(n+1, 1). \tag{c5}$$

$$\mathsf{c}_i(X_1, X_2, X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1, \ldots, m;\ j = 1, 2, 3;\ X_j = 0, 1 \tag{c5}$$

$$\mathsf{nphi} \leftarrow \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3}). \tag{c6}$$

The atoms $\mu_{ij}(X_j)$ are defined as in the proof of Theorem 5(a). To decide α, verify if the MPE with evidence psi is positive.

To prove (d), consider the Σ_4^p-complete problem $\alpha = \exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_3 \forall \mathbf{X}_4\, \phi$, where ϕ is in 3-DNF. This formula is equivalent to

$$\exists \mathbf{X}_1 \exists! \mathbf{X}_2 X_{n+1} (\neg X_{n+1} \wedge \forall \mathbf{X}_3 \exists \mathbf{X}_4 \neg\phi) \vee (X_1 \wedge \cdots \wedge X_{n+1}),$$

where $\neg\phi$ is in 3-CNF with m clauses $L_{i1} \vee L_{i2} \vee L_{i3}$, $\mathbf{X}_2 = \{X_1, \ldots, X_n\}$, and X_{n+1} is a fresh variable. Set up the program:

$$\mathsf{true}. \tag{d0}$$

$$0.5 :: \mathsf{x}(i, 1). \qquad X_i \in \mathbf{X}_1 \tag{d1}$$

$$\mathsf{x}(i, 0) \vee \mathsf{x}(i, 1). \qquad X_i \in \mathbf{X}_{1,2,3,4} \tag{d2}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(n+1, 0), \mathsf{nphi}. \tag{d3}$$

$$\mathsf{psi} \leftarrow \mathsf{x}(0, 1), \ldots, \mathsf{x}(n+1, 1). \tag{d4}$$

$$\mathsf{c}_i(X_1, X_2, X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1, \ldots, m;\ j = 1, 2, 3;\ X_j = 0, 1 \tag{d5}$$

$$x(i, 0) \leftarrow \mathsf{nphi}. \quad \mathsf{x}(i, 1) \leftarrow \mathsf{nphi}. \qquad X_i \in \mathbf{X}_3 \tag{d6}$$

$$\mathsf{nphi} \leftarrow \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3}). \tag{d7}$$

The atoms $\mu_{ij}(X_j)$ are defined as in the proof of Theorem 5(c), assuming $\mathbf{X}_5 = \emptyset$. To decide α, verify if the MPE inference with evidence psi is positive.

To prove (e), replace rule (d7) with rule (b4) (with head nphi and variables $\mathbf{X}_4$ in the symbolic set), to encode a counter over $\mathbf{X}_4$ instead of an existential quantifier. Then the MPE inference with evidence psi is positive iff $\exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_4 \#_{\geq t} \mathbf{X}_4\, \phi$ is true, where ϕ is a formula in 3-DNF.

To prove (f), replace sum with max in the program described in (e) to decide a problem $\exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_4 \forall \mathbf{X}_4\, \phi$ where ϕ is in 3-DNF.

To prove (g), consider the Σ_4^{PPP}-complete problem $\exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_4 \forall \mathbf{X}_4 \#_{<s} \mathbf{X}_5\, \phi$ where ϕ is in 3-DNF. Let $\mathbf{X}_4 = \{X_{41}, \ldots, X_{4p}\}$. Repeat the program in (c) with the appropriate changes, replacing rule (d7) with

$$\mathsf{nphi} \leftarrow \mathbf{e}(X_{41}), \ldots, \mathbf{e}(X_{4p}), \tag{g7}$$
$$\#\mathsf{sum}\{1, \mathbf{X}_5 : \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3})\} < s.$$

The formula is true iff the MPE inference with evidence psi is positive. To prove hardness for Rel(sum), replace rules (d2) with

$$\mathsf{x}(i,0) \leftarrow \#\mathsf{sum}\{-1:\mathsf{x}(i,0); 1:\mathsf{x}(i,1)\} \leq 0. \qquad X_i \in \mathbf{X}_{1,2,3} \tag{g2'}$$

$$\mathsf{x}(i,1) \leftarrow \#\mathsf{sum}\{-1:\mathsf{x}(i,0); 1:\mathsf{x}(i,1)\} \geq 0. \qquad X_i \in \mathbf{X}_{1,2,3} \tag{g2"}$$

Finally, to prove (h) replace sum with max in the previous construction to decide a problem $\exists \mathbf{X}_1 \forall \mathbf{X}_2 \exists \mathbf{X}_4 \neg \exists \mathbf{X}_4 \forall \mathbf{X}_5 \, \phi$ where ϕ is in 3-DNF. The MPE inference with evidence psi is positive iff the formula is true. □

4.3. MAP complexity

Finally, we look into the complexity of MAP inference. For positive normal propositional programs, MAP is $\mathsf{NP}^{\mathsf{PP}}$-complete [16]. The following result shows that complexity, in the propositional case, is not increased by the presence of disjunction and/or aggregates.

Theorem 9. *MAP inference is*

(a) *in* $\mathsf{NP}^{\mathsf{PP}}$ *for programs in* Prop(not, $\vee$, #), *and*
(b) *in* $\mathsf{NP}^{\mathsf{PP}^{\mathsf{PP}}}$ *for programs in* Rel(not, $\vee$, #sum, #count, #max).

Proof. (a) Guess a partial interpretation Q deciding the values for atoms indicated in $Q_1, \ldots, Q_n$, and consistent with the evidence E, and run cautious reasoning to decide whether $\underline{P}(Q, E) > \gamma$. The latter takes effort $\mathsf{PP}^{\Sigma_3^{\mathsf{p}}}$ by Theorem 3; hence the whole process is in $\mathsf{NP}^{\mathsf{PP}^{\Sigma_3^{\mathsf{p}}}}$. The result follows as $\mathsf{PP}^{\Sigma_k^{\mathsf{p}}} \subseteq \mathsf{P}^{\mathsf{PP}}$ for any k [35]; the intermediate P can be encoded into the base machine.

(b) Again, guess a partial interpretation for Q consistent with the evidence, and run cautious inference to decide whether $\underline{P}(Q, E) > \gamma$. The latter takes effort $\mathsf{PP}^{\Sigma_3^{\mathsf{p}\,\mathsf{PP}}}$ by Theorem 3. Now, we have $\Sigma_k^{p\,\mathsf{PP}} \subseteq \mathsf{BPP}^{\oplus \mathsf{P}^{\mathsf{PP}}}$ by relativizing the first part of Toda's proof [36]; hence we obtain $\mathsf{PP}^{\Sigma_k^{\mathsf{p}\,\mathsf{PP}}} \subseteq \mathsf{PP}^{\mathsf{BPP}^{\oplus \mathsf{P}^{\mathsf{PP}}}}$. The latter class is equal to $\mathsf{PP}^{\oplus \mathsf{P}^{\mathsf{PP}}}$ [35, Theorem 4.9]. We can then relativize the second part of Toda's proof [37] to obtain $\mathsf{PP}^{\oplus \mathsf{P}^{\mathsf{PP}}} \subseteq \mathsf{P}^{\mathsf{PP}^{\mathsf{PP}}}$. Thus $\mathsf{PP}^{\Sigma_k^{\mathsf{p}\,\mathsf{PP}}} \subseteq \mathsf{P}^{\mathsf{PP}^{\mathsf{PP}}}$ and the proof is completed as in item (a). □

The corresponding hardness result is:

Theorem 10. *MAP inference is*

(a) $\mathsf{NP}^{\mathsf{PP}}$*-hard for programs in* Prop(), *and*
(b) $\mathsf{NP}^{\mathsf{PP}^{\mathsf{PP}}}$*-hard for programs in* Rel(#sum).

Proof. (a) Consider the $\mathsf{NP}^{\mathsf{PP}}$-complete problem: $\psi = \exists \mathbf{X}_1 \#_{>t} \mathbf{X}_2 \, \phi$ where ϕ is in 3-CNF with clauses $L_{i1} \vee L_{i2} \vee L_{i3}$, $i = 1, \ldots, m$. Using again the argument in [34, Proposition 4], we can assume w.l.o.g. that ϕ is monotone. Hence, set up the program:

$$0.5 :: \mathsf{x}(i,1). \qquad \text{for } X_i \in \mathbf{X}_{1,2} \tag{a1}$$

$$\mathsf{c}_i \leftarrow L_{i1}. \quad \mathsf{c}_i \leftarrow L_{i2}. \quad \mathsf{c}_i \leftarrow L_{i3}. \qquad \text{for } j = 1, \ldots, m \tag{a2}$$

$$\mathsf{phi} \leftarrow \mathsf{c}_1, \ldots, \mathsf{c}_m. \tag{a3}$$

In rules (a2) L_{ij} encodes the corresponding literal in the ith clause using either $\mathsf{x}(j,1)$ or $\mathsf{x}(j,0)$. Let Q denote the indicator variables for the atoms $\mathsf{x}(i,1)$ for $X_i \in \mathbf{X}_1$, and let E be the indicator of phi. Then $\max_q \underline{P}(Q = q, E = 1) > t/2^n$ iff ψ is satisfiable, where $n = |\mathbf{X}_{1,2}|$.

(b) Consider the $\mathsf{NP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete problem $\exists \mathbf{X}_1 \#_t \mathbf{X}_2 \#_{\geq u} \mathbf{X}_3 \, \phi$, where ϕ is in 3-CNF with clauses $L_{i1} \vee L_{i2} \vee L_{i3}$, $i = 1, \ldots, m$. Set up the program:

$$\mathsf{true}. \tag{b0}$$

$$0.5 :: \mathsf{x}(i,1). \qquad X_i \in \mathbf{X}_{1,2} \tag{b1}$$

$$\mathsf{x}(i,0) \leftarrow \#\mathsf{sum}\{-1:\mathsf{x}(i,0); 1:\mathsf{x}(i,1)\} \leq 0. \qquad X_i \in \mathbf{X}_{1,2} \tag{b2}$$

$$\mathsf{c}_i(X_1, X_2, X_3) \leftarrow \mu_{ij}(X_j). \qquad i = 1, \ldots, m \tag{b3}$$

$$j = 1, 2, 3;\ X_j = 0, 1$$

$$\mathsf{phi} \leftarrow \#\mathsf{sum}\{1, \mathbf{X}_3 : \mathsf{c}_1(X_{11}, X_{12}, X_{13}), \ldots, \mathsf{c}_m(X_{m1}, X_{m2}, X_{m3})\} \geq u. \tag{b4}$$

The atoms $\mu_{ij}(X_j)$ are defined as in the proof of Theorem 5(c). The MAP problem with $Q_1, \ldots, Q_n$ being indicator variables on $\mathsf{x}(i,1)$ for $X_i \in \mathbf{X}_{1,2}$, evidence and $E = \mathsf{phi}$ and threshold $t/2^n$, where $n = |\mathbf{X}_{1,2}|$ then solves the satisfiability problem. □

5. Conclusion

We derived several new results on the complexity of probabilistic answer set programming under the credal semantics, from cautious reasoning to MPE to MAP, for propositional and bounded-arity relational programs. In particular, we analyzed the complexity when programs have aggregates, disjunctions in rule heads and integrity constraints. Our results for propositional programs mirror those for nonprobabilistic programs: aggregates provide the same computational power as disjunctions, and the complexity is not altered when one construct is added to the other; moreover, stratified aggregates behave as stratified negation. The case is more interesting when variables and bounded-arity predicates are considered. There the aggregates introduce a complexity on their own that varies with the type of aggregate used.

We note that several results in this paper offer interesting hard problems for complexity classes in the counting hierarchy; some of these classes are rarely visited in the literature. Consider, as one example, the $\mathsf{NP}^{\mathsf{PP}^{\mathsf{PP}}}$-hardness of MAP inference for relational programs with aggregates.

We left for the future the complexity analysis when weak constraints and strong negation are allowed, and when the program is fixed and the input is just the query. An analysis of the complexity of checking consistency of an input probabilistic answer set program is also left for future work.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

The first author received financial support by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 303920/2016-5 (PQ) and 420669/2016-7. The second author is partially supported by the CNPq grant 312180/2018-7 (PQ). This work has been supported in part by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), grants 2015/21880-4, 2016/18841-0, 2019/07665-4, and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – finance code 001.

References

[1] S. Muggleton, Learning stochastic logic programs, Electron. Trans. Artif. Intell. 4 (2000) 141–153.
[2] D. Poole, Abducing through negation as failure: stable models within the independent choice logic, J. Log. Program. 44 (2000) 5–35.
[3] T. Sato, Y. Kameya, N.-F. Zhou, Generative modeling with failure in PRISM, in: International Joint Conference on Artificial Intelligence, 2005, pp. 847–852.
[4] T. Lukasiewicz, Probabilistic description logic programs, Int. J. Approx. Reason. 45 (2) (2007) 288–307.
[5] S. Hadjichristodoulou, D. Warren, Probabilistic logic programming with well-founded negation, in: International Symposium on Multiple-Valued Logic, 2012, pp. 232–237.
[6] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic programs with annotated disjunctions, in: Proceedings of the International Conference on Logic Programming, 2004, pp. 431–445.
[7] S. Michels, A. Hommersom, P. Lucas, M. Velikova, A new probabilistic constraint logic programming language based on a generalised distribution semantics, Artif. Intell. 228 (2015) 1–44.
[8] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt, Inference and learning in probabilistic logic programs using weighted Boolean formulas, Theory Pract. Log. Program. 15 (3) (2015) 358–401.
[9] T. Eiter, G. Ianni, T. Krennwallner, Answer set programming: a primer, in: Tutorial Lectures of the 5th Reasoning Web International Summer School on Reasoning Web: Semantic Technologies for Information Systems, Springer-Verlag, Berlin, 2009, pp. 40–110.
[10] L. De Raedt, Logical and Relational Learning, Springer, 2008.
[11] L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton, Probabilistic Inductive Logic Programming, Springer, 2010.
[12] F. Riguzzi, E. Bellodi, R. Zese, G. Cota, E. Lamma, A survey of lifted inference approaches for probabilistic logic programming under the distribution semantics, Int. J. Approx. Reason. 80 (2017) 313–333.
[13] F. Riguzzi, Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning, River Publishers, 2018.
[14] I. Ceylan, T. Lukasiewicz, R. Peñaloza, Complexity results for probabilistic Datalog$^{\pm}$, in: Proceedings of the 22nd European Conference on Artificial Intelligence, 2016, pp. 1414–1422.
[15] F. Cozman, D. Mauá, On the semantics and complexity of probabilistic logic programs, J. Artif. Intell. Res. 60 (2017) 221–262.
[16] F. Cozman, D. Mauá, The complexity of inferences and explanations in probabilistic logic programming, in: Proceedings of the 14th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2017, pp. 449–458.
[17] T. Sato, A statistical learning method for logic programs with distribution semantics, in: International Conference on Logic Programming, 1995, pp. 715–729.
[18] W. Faber, G. Pfeifer, N. Leone, Semantics and complexity of recursive aggregates in answer set programming, Artif. Intell. 175 (2011) 278–298.
[19] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[20] K.R. Apt, H.A. Blair, A. Walker, Foundations of deductive databases and logic programming, chap. Towards a Theory of Declarative Knowledge, Morgan Kaufmann Publishers Inc., 1988, pp. 89–148.
[21] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proceedings of the Fifth International Conference on Logic Programming, 1988, pp. 1070–1080.
[22] T.C. Son, E. Pontelli, A constructive semantic characterization of aggregates in answer set programming, Theory Pract. Log. Program. 7 (3) (2007) 355–375.
[23] P. Ferraris, Logic programs with propositional connectives and aggregates, ACM Trans. Comput. Log. 12 (4) (2011) 25.
[24] M. Gelfond, Y. Zhang, Vicious circle principle and logic programs with aggregates, Theory Pract. Log. Program. 14 (4–5) (2014) 587–601.

[25] T. Eiter, W. Faber, M. Fink, S. Woltran, Complexity results for answer set programming with bounded predicate arities and implications, Ann. Math. Artif. Intell. (2007) 51–123.
[26] C. Baral, M. Gelfond, N. Rushton, Probabilistic reasoning with answer sets, Theory Pract. Log. Program. 9 (1) (2009) 57–144.
[27] 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, 2016, pp. 145–154.
[28] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[29] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comput. 6 (4) (1977) 675–695.
[30] K. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Inform. 23 (3) (1986) 325–356.
[31] J. Tóran, Complexity classes defined by counting quantifiers, J. ACM 38 (3) (1991) 753–774.
[32] L.J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1977) 1–22.
[33] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: propositional case, Ann. Math. Artif. Intell. 15 (3–4) (1995) 289–323.
[34] J. Goldsmith, M. Hagen, M. Mundhenk, Complexity of DNF minimization and isomorphism testing for monotone formulas, Inf. Comput. 206 (6) (2008) 760–775.
[35] S. Toda, PP is as hard as the polynomial-time hierarchy, SIAM J. Comput. 20 (5) (1991) 865–877.
[36] L. Fortnow, A simple proof of Toda's theorem, Theory Comput. 5 (2009) 135–140.
[37] F. Green, J. Kobler, J. Tóran, The power of the middle bit, in: Proceedings of the 7th Annual Structure in Complexity Theory Conference, 1992, pp. 111–117.