Journal of Artificial Intelligence Research 60 (2017) 221-262 Submitted 02/17; published 09/17

On the Semantics and Complexity of Probabilistic Logic Programs

Fabio Gagliardi Cozman FGCOZMAN@USP.BR
Escola Politécnica, Universidade de São Paulo, Brazil

Denis Deratani Mauá DDM@IME.USP.BR
Instituto de Matemática e Estatística,
Universidade de São Paulo, Brazil

Abstract

We examine the meaning and the complexity of probabilistic logic programs that consist of a set of rules and a set of independent probabilistic facts (that is, programs based on Sato's distribution semantics). We focus on two semantics, respectively based on stable and on well-founded models. We show that the semantics based on stable models (referred to as the "credal semantics") produces sets of probability measures that dominate infinitely monotone Choquet capacities; we describe several useful consequences of this result. We then examine the complexity of inference with probabilistic logic programs. We distinguish between the complexity of inference when a probabilistic program and a query are given (the *inferential* complexity), and the complexity of inference when the probabilistic program is fixed and the query is given (the *query* complexity, akin to *data* complexity as used in database theory). We obtain results on the inferential and query complexity for acyclic, stratified, and normal propositional and relational programs; complexity reaches various levels of the counting hierarchy and even exponential levels.

1. Introduction

The combination of deterministic and uncertain reasoning has led to many mixtures of logic and probability (Halpern, 2003; Hansen & Jaumard, 1996; Nilsson, 1986). In particular, combinations of logic programming constructs and probabilistic assessments have been pursued in several guises (De Raedt, Frasconi, Kersting, & Muggleton, 2010; De Raedt, 2008).

Among probabilistic logic programming languages, a popular strategy is to add independent probabilistic facts to a logic program — an approach that appeared in Poole's *Probabilistic Horn Abduction* (Poole, 1993), Sato's *distribution semantics* (Sato, 1995) and Fuhr's *Probabilistic Datalog* (Fuhr, 1995). For instance, consider a program with a rule

$$\mathsf{alarm} :- \mathsf{earthquake}, \mathsf{sensorOn}.$$

and suppose fact sensorOn is added to the program with probability 0.1. Depending on the presence of sensorOn, earthquake may or may not trigger alarm.

The initial efforts by Poole, Sato and Fuhr respectively focused on *acyclic* or *definite* or *stratified* programs. Indeed, Sato's distribution semantics was announced rather generally, "roughly, as distributions over least models" (Sato, 1995). Since then, there has been significant work on non-definite and on non-stratified probabilistic logic programs under variants of the distribution semantics (Hadjichristodoulou & Warren, 2012; Lukasiewicz, 2005; Riguzzi, 2015; Sato, Kameya, & Zhou, 2005).

In this paper we examine the meaning and the computational complexity of probabilistic logic programs that extend Sato's distribution semantics. We look at function-free normal programs containing negation as failure and probabilistic facts. The goal is to compute an *inference*; that is, to compute the probability $\mathbb{P}(\mathbf{Q}|\mathbf{E})$, where both $\mathbf{Q}$ and $\mathbf{E}$ refer to events assigning truth values to selected ground atoms. The pair $(\mathbf{Q}, \mathbf{E})$ is referred to as the *query*. We distinguish between the complexity of inference when a probabilistic program and a query are the input (the *inferential* complexity), and the complexity of inference when the probabilistic program is fixed and the query is the input (the *query* complexity). Query complexity is similar to *data* complexity as used in database theory, as we discuss later.

We first examine acyclic programs; for those programs all existing semantics coincide. Given the well-known relationship between acyclic probabilistic logic programs and Bayesian networks, it is not surprising that inference for propositional acyclic programs is PP-complete. However, it *is* surprising that, as we show, inference with *bounded arity* acyclic programs is $\mathsf{PP}^{\mathsf{NP}}$-complete, thus going up the counting hierarchy. And we show that acyclic programs without a bound on predicate arity take us to PEXP-completeness.

We then move to cyclic normal logic programs. Many useful logic programs are cyclic; indeed, the use of recursion is at the heart of logic programs and its various semantics (Gelfond & Lifschitz, 1988; Van Gelder, Ross, & Schlipf, 1991). Many applications, such as non-recursive structural equation models (Berry, 1984; Pearl, 2009) and models with "feedback" (Nodelman, Shelton, & Koller, 2002; Poole & Crowley, 2013), defy the acyclic character of Bayesian networks.

First we look at the inferential and query complexity of a subclass of cyclic programs known as locally stratified programs. Such programs allow one to have recursion, as long as no negation appears in cycles. For instance, we might have a rule that recursively defines the concept of a path using the predicate edge:

$$\mathsf{path}(X, Y) :- \mathsf{edge}(X, Z), \mathsf{path}(Z, Y).$$

For stratified programs, again we have that most existing semantics coincide; in particular semantics based on stable and well-founded models are identical. Moreover, stratified probabilistic logic programs share an important property with acyclic programs: each one of these programs specifies a unique probability distribution over groundings. To summarize, we show that the complexity of stratified programs is the same as the complexity of acyclic programs.

We then study general (cyclic) programs. There are various semantics for such programs, and relatively little discussion about them in the literature. For instance, suppose we have the following rules

$$\mathsf{sleep} :- \mathbf{not}\ \mathsf{work}, \mathbf{not}\ \mathsf{insomnia}. \qquad \mathsf{work} :- \mathbf{not}\ \mathsf{sleep}. \tag{1}$$

and suppose fact insomnia is added to the program with probability 0.3. Hence, with probability 0.3, we have that insomnia is true, and then sleep is false and work is true. This is simple enough. But with probability 0.7, we have that insomnia is absent, thus insomnia is false, and then the remaining two rules create a cycle: sleep depends on work and vice-versa. The question is how to define a semantics when a cycle appears.

We focus on two semantics for such programs, even though we mention a few others. The first semantics, which we call "credal semantics", is based on work by Lukasiewicz on

probabilistic description logics (Lukasiewicz, 2005, 2007). His proposal is that a probabilistic logic program defines a set of probability measures, induced by the various stable models of the underlying normal logic program. We show that credal semantics produces sets of probability models that dominate infinitely monotone Choquet capacities; the latter objects are relatively simple extensions of probability distributions and have been often used in the literature, from random set theory to Dempster-Shafer theory.

The second semantics is based on the well-founded models of normal logic programs: in this case there is always a single distribution induced by a probabilistic logic program (Hadjichristodoulou & Warren, 2012).

We show that the complexity of inference under the credal semantics goes up the counting hierarchy, up to $\mathsf{PP}^{\mathsf{NP}^{\mathsf{NP}}}$ levels, whereas the complexity under the well-founded semantics remains equivalent to that of acylic or stratified programs.

The paper begins in Section 2 with a review of logic programming and complexity theory. Section 3 presents basic notions concerning probabilistic logic programs and their semantics. Our main results appear in Sections 4, 5, 6 and 7. In Section 4 we show that the credal semantics of a probabilistic logic program is a set of probability measures induced by an infinitely monotone Choquet capacity. Sections 5, 6 and 7 analyze the complexity of inferences under the credal and the well-founded semantics. In Section 8 we contribute with a comparison between the credal and the well-founded semantics. The paper concludes, in Section 9, with a summary of our contributions and a discussion of future work.

2. Background

We briefly collect here some well known terminology and notation regarding logic programming (Sections 2.1 and 2.2) and complexity theory (Section 2.3).

Before we plunge into those topics, we briefly fix notation on Bayesian networks as we will need them later. A *Bayesian network* is a pair consisting of a directed acyclic graph $\mathbb{G}$ whose nodes are random variables, and a joint probability distribution $\mathbb{P}$ over all variables in the graph, such that $\mathbb{G}$ and $\mathbb{P}$ satisfy the "directed Markov condition" (i.e., a random variable is independent of its parents given its nondescendants) (Koller & Friedman, 2009; Neapolitan, 2003; Pearl, 1988). If all random variables are discrete, then one can specify "local" conditional probabilities $\mathbb{P}(X_i = x_i|\text{pa}(X_i) = \pi_i)$, and the joint probability distribution is necessarily the product of these local probabilities:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} \mathbb{P}(X_i = x_i|\text{pa}(X_i) = \pi_i)\,, \tag{2}$$

where π_i is the projection of $\{x_1, \ldots, x_n\}$ on the set of random variables $\text{pa}(X_i)$; whenever X_i has no parents, $\mathbb{P}(X_i = x_i|\text{pa}(X_i) = \pi_i)$ stands for $\mathbb{P}(X_i = x_i)$.

2.1 Normal Logic Programs: Syntax

Take a vocabulary consisting of a set of logical variables $X, Y, \ldots$, a set of predicate names $\mathsf{r}, \mathsf{s}, \ldots$, and a set of constants $a, b, \ldots$. Each predicate is associated with a nonnegative integer denoting its arity. An *atom* is any expression of the form $\mathsf{r}(t_1, \ldots, t_n)$, where r is a

predicate of arity n and each t_i is either a constant or a logical variable. A zero-arity atom is written simply as r. An atom is *ground* if it does not contain logical variables.

A *normal logic program* consists of rules written as (Dantsin, Eiter, & Voronkov, 2001)

$$A_0 :- A_1, \ldots, A_m, \mathbf{not} A_{m+1}, \ldots, \mathbf{not} A_n.$$

where the A_i are atoms and **not** is interpreted according to some selected semantics, as discussed later. The *head* of this rule is A_0; the remainder of the rule is its *body*. A rule without a body, written simply as $A_0.$, is a *fact*. A *subgoal* in the body is either an atom A (a *positive* subgoal) or **not** A (a *negative* subgoal). A program without negation is *definite*, and a program with only ground atoms is *propositional*.

Example 1. Here is a program describing the supposed relation between smoking, stress, and social influence (Fierens, Van den Broeck, Renkens, Shrerionov, Gutmann, Janssens, & De Raedt, 2014):

$$\mathsf{smokes}(X) :- \mathsf{stress}(X).$$
$$\mathsf{smokes}(X) :- \mathsf{influences}(Y, X), \mathsf{smokes}(Y).$$
$$\mathsf{influences}(a, b). \ \mathsf{influences}(b, a). \ \mathsf{stress}(b).$$

This program is definite, but not propositional. □

The semantics of a definite program is simply the unique minimal model of the program; programs with negation require more complex semantic notions. We review the concept of *model*, and the machinery needed to address non-definite programs, in the next section.

2.2 Normal Logic Programs: Semantics

The *Herbrand base* of a program is the set of all ground atoms built from constants and predicates in the program. We do not consider functions in this paper, to stay with finite Herbrand bases.

A *substitution* is a (partial) function that maps logical variables into terms. An atom A *unifies* with an atom B if there is a substitution that makes both (syntactically) equal. A *grounding* is a substitution mapping into constants. The grounding of a rule is a ground rule obtained by applying the same grounding to each atom. The grounding of a program is the propositional program obtained by applying every possible grounding to each rule, using only the constants in the program (i.e., using only ground atoms in the Herbrand base).

A *literal* L is either an atom A or a negated atom $\neg A$. A set of literals is *inconsistent* if A and $\neg A$ belong to it. Given a normal logic program $\mathbf{P}$, a *partial interpretation* is a consistent set of literals whose atoms belong to the Herbrand base of $\mathbf{P}$. An *interpretation* is a consistent set of literals such that every atom in the Herbrand base appears in a literal. An atom is true (resp., false) in a (partial) interpretation if it appears in a non-negated (resp., negated) literal. A subgoal is true in an interpretation if it is an atom A and A belongs to the interpretation, or if the subgoal is **not** A and $\neg A$ belongs to the interpretation. A grounded rule is *satisfied* in a partial interpretation if its head is true in the interpretation, or any of its subgoals is false in the interpretation. A *model* of $\mathbf{P}$ is an interpretation such that every grounding of a rule in $\mathbf{P}$ is satisfied. A *minimal model* of $\mathbf{P}$ is a model with minimum number of non-negated literals.

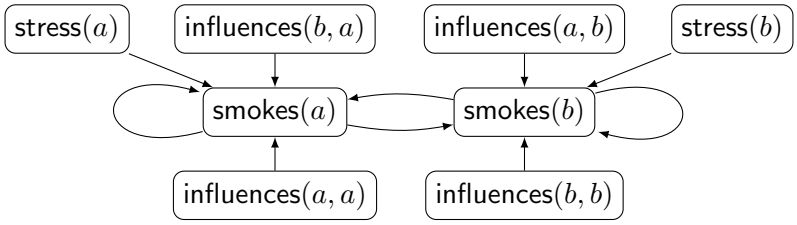

Figure 1: Grounded dependency graph for Example 1.

The *dependency graph* of a program is a directed graph where each predicate is a node, and where there is an edge from a node B to a node A if there is a rule where A appears in the head and B appears in the body; if B appears right after **not** in any such rule, the edge is *negative*; otherwise, it is *positive*. The *grounded* dependency graph is the dependency graph of the propositional program obtained by grounding. For instance, the grounded dependency graph of the program in Example 1 is depicted in Figure 1.

A program is *acyclic* when its grounded dependency graph is acyclic.

Concerning the semantics of normal logic programs, there are, broadly speaking, two strategies to follow. One strategy is to translate programs into a first-order theory that is called a *completion* of the program. Then the semantics of the program is the set of first-order models of its completion. The most famous completion is Clark's, roughly defined as follows (Clark, 1978). First, rewrite each body by replacing comma by $\wedge$ and **not** by $\neg$. Second, remove constants from heads: to do so, consider a rule $A_0(a) :- B_i\bullet$, where a is a constant and B_i is the body; then this rule is replaced by $A_0(X) :- (X = a) \wedge B_i\bullet$. Then, for each set of rules that share the same head $A_0(X)$, write $A_0(X) \Leftrightarrow B_1 \vee B_2 \vee \ldots \vee B_k$, where each B_i is the body of one of the rules (we might need to rename some of the logical variables in the bodies to make them all consistent with the head $A_0(X)$).

The second strategy that is often used to define the semantics of normal logic programs is to select some models of a program to be its semantics. There are many proposals in the literature as to which models should be selected; however, currently there are two selections that have received most attention: the *stable model* (Gelfond & Lifschitz, 1988) and the *well-founded* (Van Gelder et al., 1991) semantics. We now describe these semantics; alas, their definitions are not simple. We assume the programs are propositional; the semantics of a non-propositional program is the semantics of their grounding.

Consider first the stable model semantics. Suppose we have a normal logic program $\mathbf{P}$ and an interpretation $\mathcal{I}$. Define the *reduct* $\mathbf{P}^{\mathcal{I}}$ to be a definite program that contains rule $A_0 :- A_1, \ldots, A_m\bullet$ iff one of the grounded rules from $\mathbf{P}$ is $A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n\bullet$ where each $A_{m+1}, \ldots, A_n$ is false in $\mathcal{I}$. That is, the reduct is obtained by (i) grounding $\mathbf{P}$, (ii) removing all rules that contain a subgoal **not** A in their body such that A is an atom that is true in $\mathcal{I}$, (iii) removing all remaining literals of the form **not** A from the remaining rules. An interpretation $\mathcal{I}$ is a *stable model* iff $\mathcal{I}$ is a minimal model of $\mathbf{P}^{\mathcal{I}}$. Note that a normal program may fail to have a stable model, or may have several stable models.[1]

There are two types of logical reasoning under the stable mode semantics (Eiter, Faber, Fink, & Woltran, 2007). *Brave reasoning* asks whether there is a stable model containing

1. Other concepts of reduct are possible (Faber, Pfeifer, & Leone, 2011).

a specific atom (and possibly returns one such model if it exists). *Cautious reasoning* asks whether a specific atom appears in all stable models (and possibly lists all such models).

Consider now the well-founded semantics. Given a subset $\mathcal{U}$ of the Herbrand base of a program, and a partial interpretation $\mathcal{I}$, say that an atom A is *unfounded* with respect to $\mathcal{U}$ and $\mathcal{I}$ iff for each rule whose head is A, we have that (i) some subgoal is false in $\mathcal{I}$, or (ii) some positive subgoal is in $\mathcal{U}$. Now say that a subset $\mathcal{U}$ of the Herbrand base is an *unfounded set* with respect to interpretation $\mathcal{I}$ if each atom in $\mathcal{U}$ is unfounded with respect to $\mathcal{U}$ and $\mathcal{I}$. This is a complex definition: roughly, it means that, for each possible rule that we might apply to obtain A, either the rule cannot be used (given $\mathcal{I}$), or there is an atom in $\mathcal{U}$ that must be first shown to be true. Define $\mathbb{U}_{\mathbf{P}}(\mathcal{I})$ to be the greatest unfounded set with respect to $\mathcal{I}$ (there is always such a greatest set). Now, given normal logic program $\mathbf{P}$, define $\mathbb{T}_{\mathbf{P}}(\mathcal{I})$ to be a transformation that takes interpretation $\mathcal{I}$ and returns another interpretation: $A \in \mathbb{T}_{\mathbf{P}}(\mathcal{I})$ iff there is some rule with head A such that every subgoal in the body is true in $\mathcal{I}$. Define $\mathbb{W}_{\mathbf{P}}(\mathcal{I}) = \mathbb{T}_{\mathbf{P}}(\mathcal{I}) \cup \neg\mathbb{U}_{\mathbf{P}}(\mathcal{I})$, where the notation $\neg\mathbb{U}_{\mathbf{P}}(\mathcal{I})$ means that we take each literal in $\mathbb{U}_{\mathbf{P}}(\mathcal{I})$ and negate it (that is, A becomes $\neg A$; $\neg A$ becomes A). Intuitively, $\mathbb{T}_{\mathbf{P}}$ is what we can "easily prove to be positive" and $\mathbb{U}_{\mathbf{P}}$ is what we can "easily prove to be negative".

Finally: the well-founded semantics of $\mathbf{P}$ is the least fixed point of $\mathbb{W}_{\mathbf{P}}(\mathcal{I})$; this fixed point always exists. That is, apply $\mathcal{I}_{i+1} = \mathbb{W}_{\mathbf{P}}(\mathcal{I}_i)$, starting from $\mathcal{I}_0 = \emptyset$, until it stabilizes; the resulting interpretation is the well-founded model. The iteration stops in finitely many steps given that we have finite Herbrand bases.

The well-founded semantics determines the truth assignment for a subset of the atoms in the Herbrand base; for the remaining atoms, their "truth values are not determined by the program" (Van Gelder et al., 1991, Section 1.3, p. 623). A very common interpretation of this situation is that the well-founded semantics uses three-valued logic with values true, false, and undefined. It so happens that any well-founded model is a subset of every stable model of a normal logic program (Van Gelder et al., 1991, Corollary 5.7); hence, if a program has a well-founded model that is an interpretation for all atoms, then this well-founded model is the unique stable model (the converse is not true).

There are other ways to define the well-founded semantics that are explicitly constructive (Baral & Subrahmanian, 1993; Van Gelder, 1993; Przymusinski, 1989). One is this, where the connection with the stable model semantics is emphasized (Baral & Subrahmanian, 1993): write $\mathbb{LFT}_{\mathbf{P}}(\mathcal{I})$ to mean the least fixpoint of $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$; then the well-founded semantics of $\mathbf{P}$ consists of those atoms A that are in the least fixpoint of $\mathbb{LFT}_{\mathbf{P}}(\mathbb{LFT}_{\mathbf{P}}(\cdot))$ plus the literals $\neg A$ for those atoms A that are *not* in the greatest fixpoint of $\mathbb{LFT}_{\mathbf{P}}(\mathbb{LFT}_{\mathbf{P}}(\cdot))$. Note that $\mathbb{LFT}_{\mathbf{P}}(\mathbb{LFT}_{\mathbf{P}}(\cdot))$ is a monotone operator.

Here are three examples to which we return later:

Example 2. First, take a program $\mathbf{P}$ with the two rules in Expression (1): sleep :− **not** work, **not** insomnia**.** and work :− **not** sleep**.**. This program has two stable models: both assign false to insomnia; one assigns true to work and false to sleep, while the other assigns true to sleep and false to work. The well-founded semantics assigns false to insomnia and leaves sleep and work as undefined. □

Example 3. Consider a game where a player wins whenever there is a move to another player with no more moves (Van Gelder et al., 1991; Van Gelder, 1993), as expressed by the

cyclic rule:

$$\mathsf{wins}(X) :- \mathsf{move}(X, Y), \mathbf{not}\ \mathsf{wins}(Y).$$

Suppose the available moves are given as the following facts:

$$\mathsf{move}(a, b). \quad \mathsf{move}(b, a). \quad \mathsf{move}(b, c). \quad \mathsf{move}(c, d).$$

There are two stable models: both assign true to $\mathsf{wins}(c)$ and false to $\mathsf{wins}(d)$; one assigns true to $\mathsf{wins}(a)$ and false to $\mathsf{wins}(b)$, while the other assigns true to $\mathsf{wins}(b)$ and false to $\mathsf{wins}(a)$. The well-founded semantics leads to partial interpretation $\{\mathsf{wins}(c), \neg\mathsf{wins}(d)\}$, leaving $\mathsf{wins}(a)$ and $\mathsf{wins}(b)$ as undefined. If $\mathsf{move}(a, b)$ is not given as a fact, it is assigned false, and the well-founded semantics leads to $\{\neg\mathsf{wins}(a), \mathsf{wins}(b), \mathsf{wins}(c), \neg\mathsf{wins}(d)\}$. □

Example 4. The Barber Paradox: If the barber shaves all, and only, those villagers who do not shave themselves, does the barber shave himself? Consider:

$$\begin{array}{c} \mathsf{shaves}(X, Y) :- \mathsf{barber}(X), \mathsf{villager}(Y), \mathbf{not}\ \mathsf{shaves}(Y, Y). \\ \mathsf{villager}(a). \quad \mathsf{barber}(b). \quad \mathsf{villager}(b). \end{array} \tag{3}$$

There is no stable model for this normal logic program: the facts and the rule lead to the pattern $\mathsf{shaves}(b, b) :-$ **not** $\mathsf{shaves}(b, b)$., thus eliminating any possible stable model. The well-founded semantics assigns false to $\mathsf{barber}(a)$, to $\mathsf{shaves}(a, a)$ and to $\mathsf{shaves}(a, b)$. Also, $\mathsf{shaves}(b, a)$ is assigned true, and $\mathsf{shaves}(b, b)$ is left undefined. That is, even though the semantics leaves the status of the barber as undefined, it does produce meaningful answers for other villagers. □

2.3 Complexity Theory: The Counting Hierarchy

We adopt basic terminology and notation from computational complexity (Papadimitriou, 1994). A *language* is a set of strings; we assume all inputs are bitstrings encoding programs, sets, rational numbers (input encodings, and their sizes, are briefly described in Section 5). A language defines a *decision problem*; that is, the problem of deciding whether an input string is in the language. A *complexity class* is a set of languages; we use well-known complexity classes such as P, NP, EXP, NEXP. We often make no distinction between a language and its associated decision problem; and we refer to complexity classes as classes of decision problems. The complexity class PP consists of those languages $\mathcal{L}$ that satisfy the following property: there is a polynomial time nondeterministic Turing machine M such that $\ell \in \mathcal{L}$ iff more than half of the computations of M on input ℓ end up accepting. Analogously, we have PEXP, consisting of those languages $\mathcal{L}$ with the following property: there is an exponential time nondeterministic Turing machine M such that $\ell \in \mathcal{L}$ iff half of the computations of M on input ℓ end up accepting (Buhrman, Fortnow, & Thierauf, 1998).

An oracle Turing machine $M^{\mathcal{L}}$, where $\mathcal{L}$ is a language, is a Turing machine that can write a string ℓ to an "oracle" tape and obtain from the oracle, in unit time, the decision as to whether $\ell \in \mathcal{L}$ or not. Similarly, for a function f, an oracle Turing machine M^f can be defined. If A is a class of languages/functions that are defined by a set of Turing machines $\mathcal{M}$ (that is, the languages/functions are decided/computed by these machines), then $\mathsf{A}^{\mathcal{L}}$ is the set of languages/functions that are decided/computed by the oracle machines $M^{\mathcal{L}}$, for each M in $\mathcal{M}$. Similarly, for any class A we have A^f. If A and B are classes of languages/functions,

$\mathsf{A}^{\mathsf{B}} = \cup_{x \in \mathsf{B}} \mathsf{A}^x$. The *polynomial hierarchy* consists of classes $\Pi_i^{\mathsf{P}} = \mathsf{co}\Sigma_i^{\mathsf{P}}$, $\Sigma_i^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}}$ and $\Delta_i^{\mathsf{P}} = \mathsf{P}^{\Sigma_{i-1}^{\mathsf{P}}}$, with $\Sigma_0^{\mathsf{P}} = \mathsf{P}$. The class $\mathsf{PH} = \cup_i \Sigma_i^{\mathsf{P}}$ contains every language is the polynomial hierarchy.

Wagner's *polynomial counting hierarchy* is the smallest set of classes containing P and, recursively, for any class C in the polynomial counting hierarchy, the classes PP^{C}, NP^{C}, and $\mathsf{coNP}^{\mathsf{C}}$ (Wagner, 1986, Thm. 4; Tóran, 1991, Thm. 4.1). The polynomial hierarchy is included in Wagner's counting polynomial hierarchy.

A *many-one reduction* from $\mathcal{L}$ to $\mathcal{L}'$ is a polynomial time algorithm that takes the input to decision problem $\mathcal{L}$ and transforms it into the input to decision problem $\mathcal{L}'$ such that $\mathcal{L}'$ has the same output as $\mathcal{L}$. For a complexity class C, a decision problem $\mathcal{L}$ is C-hard with respect to many-one reductions if each decision problem in C can be reduced to $\mathcal{L}$ with many-one reductions. A decision problem is then C-complete with respect to many-one reductions if it is in C and it is C-hard with respect to many-one reductions.

In proofs we will often use propositional formulas; such a formula is in Conjunctive Normal Form (CNF) when it is a conjunction of clauses (where a clause is a disjunction of literals). A kCNF is a CNF in which each clause has k literals. We use the following $\mathsf{PP}^{\Sigma_k^{\mathsf{P}}}$-complete problem (Wagner, 1986, Thm. 7), that we refer to as $\#_k\mathsf{3CNF}(>)$:

Input: A pair (ϕ, M), where $\phi(\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_k)$ is a propositional formula in 3CNF and each $\mathbf{X}_i$ is a tuple of propositional variables, and M is an integer.

Output: Whether or not the number of truth assignments for $\mathbf{X}_0$ that satisfy the formula

$$Q_1\mathbf{X}_1 : Q_2\mathbf{X}_2 : \ldots \exists \mathbf{X}_k : \phi(\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_k),$$

is strictly larger than M, where the quantifiers alternate and each propositional variable not in $\mathbf{X}_0$ is bound to a quantifier.

A formula is in Disjunctive Normal Form (DNF) if it is a disjunction of conjunctions. Another $\mathsf{PP}^{\Sigma_k^{\mathsf{P}}}$-complete problem, referred to as $\#_k\mathsf{DNF}(>)$, is:

Input: A pair (ϕ, M), where $\phi(\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_k)$ is a propositional formula in DNF and each $\mathbf{X}_i$ is a tuple of propositional variables, and M is an integer.

Output: Whether or not the number of truth assignments for $\mathbf{X}_0$ that satisfy the formula

$$Q_1\mathbf{X}_1 : Q_2\mathbf{X}_2 : \ldots \forall \mathbf{X}_k : \phi(\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_k),$$

is strictly larger than M, where the quantifiers alternate and each propositional variable not in $\mathbf{X}_0$ is bound to a quantifier.

A detail to note is that Wagner defines a $\mathsf{PP}^{\Sigma_k^{\mathsf{P}}}$-complete problem using "$\geq M$" instead of "$> M$", but the former is equivalent to "$> M - 1$", so both inequalities can be used. The fact that we can use any M and yet resort to the class PP (where the threshold is always 1/2 of computations) is central to the theory of counting complexity (Simon, 1975, Thm. 4.4).

3. Probabilistic Normal Logic Programs

In this paper we focus on a particularly simple combination of logic programming and probabilities (Poole, 1993, 2008; Sato, 1995; Sato & Kameya, 2001), that we review in this section. The syntax we need is rather simple, so most of the discussion focuses on the semantics: Section 3.1 examines the semantics of probabilistic facts, while Sections 3.2 and 3.3 examine the semantics of various types of probabilistic programs.

A *probabilistic logic program*, abbreviated PLP, is a pair $\langle \mathbf{P}, \mathbf{PF} \rangle$ consisting of a normal logic program $\mathbf{P}$ and a set of *probabilistic facts* $\mathbf{PF}$. A probabilistic fact is a pair consisting of an atom A and a probability value α; we use the notation $\alpha :: A.$ borrowed from the ProbLog package (Fierens et al., 2014).[2] We assume that every probability value is a rational number.

Example 5. Here is a syntactically correct ProbLog program:

$$
\begin{array}{l}
0.7 :: \mathsf{burglary}. \quad 0.2 :: \mathsf{earthquake}. \\
\mathsf{alarm} :- \mathsf{burglary}, \mathsf{earthquake}, \mathsf{a1}. \\
\mathsf{alarm} :- \mathsf{burglary}, \mathbf{not}\ \mathsf{earthquake}, \mathsf{a2}. \\
\mathsf{alarm} :- \mathbf{not}\ \mathsf{burglary}, \mathsf{earthquake}, \mathsf{a3}. \\
0.9 :: \mathsf{a1}. \quad 0.8 :: \mathsf{a2}. \quad 0.1 :: \mathsf{a3}. \\
\mathsf{calls}(X) :- \mathsf{alarm}, \mathsf{neighbor}(X). \\
\mathsf{neighbor}(a). \quad \mathsf{neighbor}(b).
\end{array}
$$

There are four rules, two facts, and five probabilistic facts. □

We say that a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ is acyclic (resp., definite, stratified, etc) if the corresponding logic program $\mathbf{P}$ is acyclic (resp., definite, stratified, etc.).

3.1 The Semantics of Probabilistic Facts

The interpretation of probabilistic facts requires some pause. We assume in this discussion that atoms are ground; the semantics of PLPs with logical variables is the semantics of its grounding. Suppose we have a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ with n ground probabilistic facts (which may be groundings of probabilistic facts containing logical variables). From $\langle \mathbf{P}, \mathbf{PF} \rangle$ we can generate 2^n normal logic programs: for each probabilistic fact $\alpha :: A.$, we either include (with probability α) or not (with probability $1 - \alpha$) the fact $A.$ in $\mathbf{P}$. These choices (whether to include or not each probabilistic fact) are assumed independent: this is Sato's *independence assumption*.

For instance, consider the PLP:

$$0.5 :: \mathsf{r}. \qquad 0.5 :: \mathsf{s}. \qquad \mathsf{v} :- \mathsf{r}, \mathsf{s}. \tag{4}$$

We have four ways to write a normal logic program out of this PLP; that is, r can be selected or discarded, and likewise for s. All these normal logic programs are obtained with the same probability 0.25, and in only one of them v is true; consequently, $\mathbb{P}(\mathsf{v} = \mathsf{true}) = 0.25$.

A *total choice* θ for the PLP is a subset of the set of grounded probabilistic facts. We interpret θ as the set of facts that have been probabilistically selected to be included in $\mathbf{P}$;

2. Available at https://dtai.cs.kuleuven.be/problog/index.html. ProbLog additionally has "probabilistic rules" but those are simply syntactic sugar that we do not need here.

all other facts obtained from probabilistic facts are to be discarded. The probability of a total choice is easily computed: it is a product over the probabilities of probabilistic facts, where probabilistic fact $\alpha :: A.$ contributes with factor α if $A.$ is selected, and contributes with factor $(1-\alpha)$ if $A.$ is discarded. Now for each total choice θ we obtain a normal logic program, that we denote by $\langle \mathbf{P}, \theta \rangle$.

Example 6. The PLP in Expression (4) has two probabilistic facts, leading to four total choices, each with probability 0.25. Now consider a more complicated PLP:

$$0.5 :: \mathsf{r}. \qquad 0.6 :: \mathsf{r}. \qquad 0.2 :: \mathsf{s}(a). \qquad 0.3 :: \mathsf{s}(X). \qquad \mathsf{v} :- \mathsf{r}, \mathsf{s}(a), \mathsf{s}(b).$$

There are five ground probabilistic facts (after grounding $\mathsf{s}(X)$ appropriately); hence there are 32 total choices. Suppose we include r from the first probabilistic fact, and discard all the other probabilistic facts. This total choice has probability $0.5 \times 0.4 \times 0.8 \times 0.7 \times 0.7$; then we obtain

$$\mathsf{r}. \qquad \mathsf{v} :- \mathsf{r}, \mathsf{s}(a), \mathsf{s}(b).,$$

a program with a single stable model where r is the only true atom. By going through all possible total choices, we have that $\mathbb{P}(\mathsf{r} = \mathsf{true}) = 0.8$ (as $\mathsf{r}.$ is kept in the program by a first choice with probability 0.5 or by a second choice with probability 0.6, hence $0.5 + 0.6 - 0.5 \times 0.6 = 0.8$). Similarly, $\mathbb{P}(\mathsf{s}(a) = \mathsf{true}) = 0.2 + 0.3 - 0.2 \times 0.3 = 0.44$; note however that $\mathbb{P}(\mathsf{s}(b) = \mathsf{true}) = 0.3$. And finally, $\mathbb{P}(\mathsf{v} = \mathsf{true}) = 0.8 \times 0.44 \times 0.3 = 0.1056$. □

Sato assumes that no probabilistic fact unifies with the head of a non-fact rule (that is, a rule with a nonempty body); this is called the *disjointness condition* (Sato, 1995). From a modeling perspective this is a convenient assumption even though we do not need it in our complexity results. In fact from a modeling perspective an even stronger disjointness condition makes sense: no probabilistic fact should unify with the head of any rule (with a body or not), nor with any other probabilistic fact. Under this assumption, the probabilistic fact $\alpha :: A.$ can be directly interpreted as a probabilistic assessment $\mathbb{P}(A = \mathsf{true}) = \alpha$. Again, we do not need such an assumption for our results, but our examples will always satisfy it, and it makes sense to assume that it will always be adopted in practice.

3.2 The Semantics of Definite/Acyclic/Stratified Probabilistic Logic Programs

We can now discuss the semantics of a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$. For each total choice θ we obtain the normal logic program $\langle \mathbf{P}, \theta \rangle$. Hence the distribution over total choices induces a distribution over normal logic programs.

A common assumption is that, for each total choice θ, the resulting normal logic program $\langle \mathbf{P}, \theta \rangle$ yields a single model (Fierens et al., 2014). For instance, if $\mathbf{P}$ is definite, then $\langle \mathbf{P}, \theta \rangle$ is definite for any θ, and $\langle \mathbf{P}, \theta \rangle$ has a unique stable model that is also its unique well-founded model. Thus the unique distribution over total choices becomes a unique distribution over stable/well-founded models. This distribution is exactly Sato's *distribution semantics* (Sato, 1995). Similarly, suppose that $\mathbf{P}$ is acyclic; then $\langle \mathbf{P}, \theta \rangle$ is acyclic for any θ, and $\langle \mathbf{P}, \theta \rangle$ has a unique stable model that is also its unique well-founded model (Apt & Bezem, 1991).

Poole's and Sato's original work focused respectively on acyclic and definite programs; in both cases the semantics of resulting normal logic programs is uncontroversial. The same

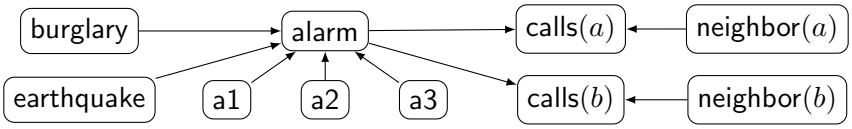

Figure 2: The grounded dependency graph for Example 5.

can be said of the larger class of *stratified* programs; a normal logic program is stratified when cycles in the grounded dependency graph contain no negative edge (this is often referred to as *locally stratified* in the literature) (Apt, Blair, & Walker, 1988). Both the stable and the well-founded semantics are identical for stratified programs, and both generate a unique interpretation for all atoms. As a consequence, a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ has a unique distribution semantics whenever $\mathbf{P}$ is stratified. Note that both acyclic and definite programs are stratified.

Example 7. The PLP in Example 5 is acyclic, and thus stratified, but not definite. The grounded dependency graph of this program is depicted in Figure 2. This graph can be interpreted as a Bayesian network, as we discuss later (Poole, 1993). There are 2^5 total choices, and the probability of $\mathsf{calls}(a)$ is 0.58.

Example 8. Consider a probabilistic version of the "smokers" program in Example 1 (Fierens et al., 2014):

$$\mathsf{smokes}(X) :- \mathsf{stress}(X).$$
$$\mathsf{smokes}(X) :- \mathsf{influences}(Y, X), \mathsf{smokes}(Y).$$
$$0.3 :: \mathsf{influences}(a, b). \quad 0.3 :: \mathsf{influences}(b, a). \quad 0.2 :: \mathsf{stress}(b).$$

The grounded dependency graph of this program is identical to the one shown in Figure 1. It is tempting to interpret this graph as a Bayesian network, but of course this is not quite right as the graph is cyclic. Indeed the program is not acyclic, but it is definite and therefore stratified, hence a unique distribution is defined over ground atoms. For instance, we have $\mathbb{P}(\mathsf{smokes}(a) = \mathsf{true}) = 0.06$ and $\mathbb{P}(\mathsf{smokes}(b) = \mathsf{true}) = 0.2$. The program would still be stratified if the first rule were replaced by $\mathsf{smokes}(X) :-$ **not** $\mathsf{stress}(X)$.. In this case there would still be a cycle, but the negative edge in the dependency graph would not belong to the cycle. □

3.3 The Semantics of General Probabilistic Logic Programs

If a normal logic program is non-stratified, then its well-founded semantics may be a partial interpretation, and some atoms may be left as undefined; it may have several stable models, or no stable model at all. Thus we must accommodate these cases when we contemplate non-stratified PLPs.

3.3.1 The Credal Semantics

A first possible semantics for general probabilistic logic programs can be extracted from work by Lukasiewicz (2005, 2007) on probabilistic description logic programs. To describe that proposal, a few definitions are needed. A PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ is *consistent* if there is at least one

stable model for each total choice of $\mathbf{PF}$. A *probability model* for a consistent PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ is a probability measure $\mathbb{P}$ over interpretations of $\mathbf{P}$, such that:

(i) every interpretation $\mathcal{I}$ with $\mathbb{P}(\mathcal{I}) > 0$ is a stable model of $\langle \mathbf{P}, \theta \rangle$ for the total choice θ that agrees with $\mathcal{I}$ on the probabilistic facts (that is, θ induces the same truth values as $\mathcal{I}$ for the grounded probabilistic facts); and

(ii) the probability of each total choice θ is the product of the probabilities for all individual choices in θ.

The set of all probability models for a PLP is the semantics of the program. Later examples will clarify this construction.

Lukasiewicz calls his proposed semantics the *answer set semantics* for probabilistic description logic programs; however, note that this name is both too restrictive (the semantics can be used for programs with functions, for instance) and a bit opaque (it does not emphasize the fact that it deals with uncertainty). We prefer the term *credal semantics*, which we adopt from now on. The reason for this latter name is that a set of probability measures is often called a *credal set* (Augustin, Coolen, de Cooman, & Troffaes, 2014).

Now given a consistent PLP, we may be interested in the smallest possible value of $\mathbb{P}(\mathbf{Q})$, where $\mathbf{Q}$ is an event assigning truth values to selected ground atoms, with respect to the set $\mathbb{K}$ of all probability models of the PLP. This is conveyed by the *lower probability* of $\mathbf{Q}$, $\underline{\mathbb{P}}(\mathbf{Q}) = \inf_{\mathbb{P} \in \mathbb{K}} \mathbb{P}(\mathbf{Q})$. Similarly, we have the *upper probability* of $\mathbf{Q}$, $\overline{\mathbb{P}}(\mathbf{Q}) = \sup_{\mathbb{P} \in \mathbb{K}} \mathbb{P}(\mathbf{Q})$. Suppose that we also have a set $\mathbf{E}$, also assigning truth values to some selected ground atoms; then we may be interested in the conditional lower and upper probabilities, respectively $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = \inf_{\mathbb{P} \in \mathbb{K}: \mathbb{P}(\mathbf{E})>0} \mathbb{P}(\mathbf{Q}|\mathbf{E})$ and $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = \sup_{\mathbb{P} \in \mathbb{K}: \mathbb{P}(\mathbf{E})>0} \mathbb{P}(\mathbf{Q}|\mathbf{E})$. We leave conditional lower/upper probabilities undefined when $\overline{\mathbb{P}}(\mathbf{E}) = 0$ (that is, when $\mathbb{P}(\mathbf{E}) = 0$ for every probability model). This is not the only possible convention: Lukasiewicz (2005, Section 3) adopts $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = 1$ and $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = 0$ in this case, while Walley's style of conditioning prescribes $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = 0$ and $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = 1$ whenever $\overline{\mathbb{P}}(\mathbf{E}) = 0$ (Walley, 1991).

3.3.2 The Well-Founded Semantics

The approach by Hadjichristodoulou and Warren (2012) is to allow probabilities directly over well-founded models, thus allowing probabilities over atoms that are undefined. That is, given a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$, associate with each total choice θ the unique well-founded model of $\langle \mathbf{P}, \theta \rangle$. The unique distribution over total choices induces a unique distribution over well-founded models (or, if one prefers, a unique distribution over three-valued truth assignments). As we discuss in Section 8, this is a bold proposal whose interpretation is far from simple.

Regardless of its meaning, the approach deserves attention as it is the only one in the literature that genuinely combines well-founded semantics with probabilities. Accordingly, we refer to it as the *well-founded semantics* of probabilistic logic programs. The combination of syntax and semantics is named WF-PRISM by Hadjichristodoulou and Warren (2012).

3.3.3 A Few Examples

Here we present three examples, with PLPs that parallel the ones discussed at the end of Section 2.2.

Example 9. Consider a probabilistic version of Example 2 (recall Expression (1)):

$$\mathsf{sleep} :- \textbf{not}\ \mathsf{work}, \textbf{not}\ \mathsf{insomnia}. \qquad \mathsf{work} :- \textbf{not}\ \mathsf{sleep}. \qquad \alpha :: \mathsf{insomnia}.$$

To interpret the PLP, note that with probability α we obtain the normal logic program

$$\mathsf{sleep} :- \textbf{not}\ \mathsf{work}, \textbf{not}\ \mathsf{insomnia}. \qquad \mathsf{work} :- \textbf{not}\ \mathsf{sleep}. \qquad \mathsf{insomnia}.$$

The unique stable/well-founded model of this program assigns true to $\mathsf{insomnia}$ and work, and false to sleep. We have the stable model $s_1 = \{\neg\mathsf{sleep}, \mathsf{work}, \mathsf{insomnia}\}$. On the other hand, with probability $1-\alpha$ we obtain a program consisting only of the two rules, with well-founded model where $\mathsf{insomnia}$ is false and both sleep and work are $\mathsf{undefined}$, and with two stable models: $s_2 = \{\mathsf{sleep}, \neg\mathsf{work}, \neg\mathsf{insomnia}\}$ and $s_3 = \{\neg\mathsf{sleep}, \mathsf{work}, \neg\mathsf{insomnia}\}$.

Consider the credal semantics. There is a probability model such that $\mathbb{P}(s_2) = 1-\alpha$ and $\mathbb{P}(s_3) = 0$, and another probability model such that $\mathbb{P}(s_2) = 0$ and $\mathbb{P}(s_3) = 1-\alpha$. Indeed any probability measure such that $\mathbb{P}(s_1) = \alpha$ and $\mathbb{P}(s_2) = \gamma(1-\alpha)$, $\mathbb{P}(s_3) = (1-\gamma)(1-\alpha)$, for $\gamma \in [0,1]$, is also a probability model for this PLP.

The well-founded semantics of the PLP is a single distribution that assigns $\mathbb{P}(s_1) = \alpha$, and assigns probability mass $1-\alpha$ to the partial interpretation $\{\neg\mathsf{insomnia}\}$.

Now consider an inference; say for instance one wants $\mathbb{P}(\mathsf{insomnia} = \mathsf{true})$. Clearly $\mathbb{P}(\mathsf{insomnia} = \mathsf{true}) = \alpha$, regardless of the semantics. But consider sleep. With respect to the credal semantics, the relevant quantities are $\underline{\mathbb{P}}(\mathsf{sleep} = \mathsf{true}) = 0$ and $\overline{\mathbb{P}}(\mathsf{sleep} = \mathsf{true}) = 1-\alpha$. And with respect to the well-founded semantics we have instead $\mathbb{P}(\mathsf{sleep} = \mathsf{true}) = 0$ and $\mathbb{P}(\mathsf{sleep} = \mathsf{false}) = \alpha$, while $\mathbb{P}(\mathsf{sleep} = \mathsf{undefined}) = 1-\alpha$. □

Example 10. Take the normal logic program discussed in Example 3, and consider the following probabilistic version (there is one probabilistic move in the game):

$$\mathsf{wins}(X) :- \mathsf{move}(X,Y), \textbf{not}\ \mathsf{wins}(Y).$$
$$\mathsf{move}(a,b). \quad \mathsf{move}(b,a). \quad \mathsf{move}(b,c). \quad 0.3 :: \mathsf{move}(\mathsf{c},d).$$

If $\mathsf{move}(c,d)$ is discarded, there is a single stable model (where b is the only winning position); otherwise, there are two stable models ($\mathsf{wins}(c)$ is true and $\mathsf{wins}(d)$ is false in both of them; $\mathsf{wins}(a)$ is true in one, while $\mathsf{wins}(b)$ is true in the other). Thus the credal semantics yields $\underline{\mathbb{P}}(\mathsf{wins}(b) = \mathsf{true}) = 0.7$ and $\overline{\mathbb{P}}(\mathsf{wins}(b) = \mathsf{true}) = 1.0$; $\underline{\mathbb{P}}(\mathsf{wins}(c) = \mathsf{true}) = 0.3$ and $\overline{\mathbb{P}}(\mathsf{wins}(c) = \mathsf{true}) = 0.3$.

Now if $\mathsf{move}(c,d)$ is discarded, the well-founded model is the unique stable model where b is the only winning position; otherwise, the well-founded model assigns true to $\mathsf{wins}(c)$ and false to $\mathsf{wins}(d)$, leaving both $\mathsf{wins}(a)$ and $\mathsf{wins}(b)$ as $\mathsf{undefined}$. Hence the well-founded semantics yields $\mathbb{P}(\mathsf{wins}(c) = \mathsf{true}) = 0.3$ and $\mathbb{P}(\mathsf{wins}(c) = \mathsf{false}) = 0.7$, while $\mathbb{P}(\mathsf{wins}(b) = \mathsf{true}) = 0.7$ and $\mathbb{P}(\mathsf{wins}(b) = \mathsf{undefined}) = 0.3$. □

Example 11. Return to the Barber Paradox discussed in Example 4, now with a probabilistic twist:

$$\mathsf{shaves}(X,Y) :- \mathsf{barber}(X), \mathsf{villager}(Y), \textbf{not}\ \mathsf{shaves}(Y,Y).$$
$$\mathsf{villager}(a). \quad \mathsf{barber}(b). \quad 0.5 :: \mathsf{villager}(b).$$

This program does not have a stable model when $\mathsf{villager}(b)$ is a fact. Thus the PLP fails to have a credal semantics.

However, the well-founded semantics is clear even when $\mathsf{villager}(b)$ is true: in this case, $\mathsf{barber}(a)$, $\mathsf{shaves}(a,a)$ and $\mathsf{shaves}(a,b)$ are false, while $\mathsf{shaves}(b,a)$ is true and $\mathsf{shaves}(b,b)$ is $\mathsf{undefined}$. And the well-founded semantics is also clear when $\mathsf{villager}(b)$ is discarded (that is, when $\mathsf{villager}(b)$ is false): only $\mathsf{shaves}(b,a)$ is true. Hence we obtain $\mathbb{P}(\mathsf{shaves}(b,a) = \mathsf{true}) = 1$, while $\mathbb{P}(\mathsf{shaves}(b,b) = \mathsf{false}) = 0.5$ and $\mathbb{P}(\mathsf{shaves}(b,b) = \mathsf{undefined}) = 0.5$. □

3.3.4 Other Semantics

Sato, Kameya and Zhou (2005) propose a semantics where distributions are defined over models produced by Fitting's three-valued semantics. We note that Fitting's semantics is weaker than the well-founded semantics, and the literature on logic programming has consistently preferred the latter, as we do in this paper.

Another three-valued approach, proposed by Lukasiewicz (2005, 2007), leaves the probability of any formula as $\mathsf{undefined}$ whenever the formula is $\mathsf{undefined}$ for some total choice (to determine whether a formula is $\mathsf{undefined}$ or not in a particular partial interpretation, three-valued logic is used). Hence, when a formula gets a (non-$\mathsf{undefined}$) numeric probability value, its truth value is the same for all stable models; thus any numeric probability calculations that are produced with this semantics agree with the semantics based on stable models (Lukasiewicz, 2007, Thm. 4.5). That is, Lukasiewicz' proposal is more akin to the credal semantics than to the well-founded semantics.

A different semantics for non-stratified PLPs is adopted by the P-log language (Baral, Gelfond, & Rushton, 2009). P-log allows for disjunction in heads and other features, but when restricted to normal logic programs it is syntactically similar to ProbLog. The semantics of a P-log program is given by a single probability distribution over possibly many stable models; whenever necessary default assumptions are called to distribute probability evenly, or to avoid inconsistent realizations (by re-normalization). A similar effect is obtained by Lee and Wang (2015) in their language $\mathrm{LP}^{\mathrm{MLN}}$, in the sense that a distribution is imposed over the various stable models — this is obtained by resorting to weighted rules that are interpreted according to Markov logic (Richardson & Domingos, 2006). We leave an analysis of this sort of semantics to the future; here we prefer to focus on semantics that do not make default assumptions concerning probabilities.

Another relevant language is CP-logic (Vennekens, Denecker, & Bruynoogue, 2009). On a syntactic level, CP-logic reminds one of ProbLog, as it adds probabilistic assessments to disjunctive logic programming. The semantics of CP-logic differs from ProbLog's, and is related to the well-founded semantics; in particular, the authors interpret the $\mathsf{undefined}$ value as "still subject to change" (akin to a marker indicating "lack of proof"). However, CP-logic adopts a *temporal precedence assumption* that can eliminate $\mathsf{undefined}$ atoms and thus force probabilistic programs to be associated with a unique probability. Programs where the temporal precedence assumption cannot be used are deemed *invalid*.

It is also important to mention the constraint logic programming language of Michels, Hommersom, Lucas, and Velikova (2015), a significant contribution that is also based on credal sets. However, they use a syntax and semantics that is markedly different from Lukasiewicz's approach, as they allow continuous variables but do not let a program have

multiple stable models per total choice. They also derive expressions for (conditional) lower and upper probabilities, by direct optimization; in Section 4 we show that such expressions can be derived from properties of infinitely monotone Choquet capacities.

Finally, Ceylan, Lukasiewicz, and Peñaloza (2016) have introduced a semantics that allows for inconsistent PLPs to have meaning without getting into three-valued logic. Instead, they use inconsistency handling techniques that automatically "repair" a program that is inconsistent under some particular context. They also adopt a much more sophisticated family of logic programs (within the Datalog$^{\pm}$ language), and they provide a thorough analysis of complexity that we discuss later. This is also a proposal that deserves future study.

We focus on the *credal* and the *well-founded* semantics in the remainder of this paper, but certainly there are other avenues to explore.

4. The Structure of the Credal Semantics

Given the generality of PLPs, one might think that credal sets generated by the credal semantics could have an arbitrarily complex structure. Surprisingly, the structure of the credal semantics of a PLP is a relatively simple object; this section is devoted to the study of this result and its consequences.

The main result of this section is:

Theorem 12. *Given a consistent* PLP*, its credal semantics is the set of all probability measures that dominate an infinitely monotone Choquet capacity.*

Before we present a proof of this theorem, let us pause and define a few terms. An infinitely monotone Choquet capacity is a set function $\underline{\mathbb{P}}$ from an algebra $\mathcal{A}$ on a set Ω to the real interval $[0, 1]$ such that (Augustin et al., 2014, Definition 4.2): $\underline{\mathbb{P}}(\Omega) = 1 - \underline{\mathbb{P}}(\emptyset) = 1$ and, for any $A_1, \ldots, A_n$ in the algebra, $\underline{\mathbb{P}}(\cup_i A_i) \geq \sum_{J \subseteq \{1,\ldots,n\}} (-1)^{|J|+1} \underline{\mathbb{P}}(\cap_{j \in J} A_j)$. Recall that an algebra is a collection of sets, including the empty set, that is closed under complement and union (we do not need infinite unions here as we can assume our sets to be finite).

Infinitely monotone Choquet capacities appear in several formalisms; for instance, they are the *belief functions* of Dempster-Shafer theory (Shafer, 1976), summaries of *random sets* (Molchanov, 2005), and *inner measures* (Fagin & Halpern, 1991).

Given an infinitely monotone Choquet capacity $\underline{\mathbb{P}}$, we can construct a set of measures that *dominate* $\underline{\mathbb{P}}$; this is the set $\{\mathbb{P} : \forall A \in \mathcal{A} : \mathbb{P}(A) \geq \underline{\mathbb{P}}(A)\}$. We abuse language and say that a set consisting of all measures that dominate an infinitely monotone Choquet capacity is an *infinitely monotone credal set*. If a credal set $\mathbb{K}$ is infinitely monotone, then the *lower probability* $\underline{\mathbb{P}}$, defined as $\underline{\mathbb{P}}(A) = \inf_{\mathbb{P} \in \mathbb{K}} \mathbb{P}(A)$, is exactly the generating infinitely monotone Choquet capacity. We also have the *upper probability* $\overline{\mathbb{P}}(A) = \sup_{\mathbb{P} \in \mathbb{K}} \mathbb{P}(A) = 1 - \underline{\mathbb{P}}(A^c)$ (where the superscript c denotes complement).

Proof of Theorem 12. Consider a set Θ containing as states the posssible total choices of the PLP. Over this space we have a product measure that is completely specified by the probabilities attached to probabilistic facts. Now consider a multi-valued mapping Γ between Θ and the space Ω of all possible models of our probabilistic logic program. For each element $\theta \in \Theta$, define $\Gamma(\theta)$ to be the set of stable models associated with the total choice θ of the

probabilistic facts. Now we use the fact that a probability space and a multi-valued mapping induce an infinitely monotone Choquet capacity over the range of the mapping (that is, over Ω) (Molchanov, 2005). Such a capacity can be equivalently represented by the set of all probability measures that dominate it (Augustin et al., 2014). □

Infinitely monotone credal sets have several useful properties; for one thing they are closed and convex. Convexity here means that if $\mathbb{P}_1$ and $\mathbb{P}_2$ are in the credal set, then $\alpha\mathbb{P}_1 + (1-\alpha)\mathbb{P}_2$ is also in the credal set for $\alpha \in [0,1]$. Thus, as illustrated by Example 9. Thus we have:

Corollary 13. *Given a consistent* PLP*, its credal semantics is a closed and convex set of probability measures.*

There are several additional results concerning the representation of infinitely monotone capacities using their Möbius transforms (Augustin et al., 2014; Shafer, 1976); we refrain from mentioning every possible corollary we might produce here. Instead, we focus on a few important results that can be used to great effect in future applications. First, as we have a finite Herbrand base, we can use the symbols in the proof of Theorem 12 to write, for any set of stable models $\mathcal{M}$ (Augustin et al., 2014, Section 5.3.2):

$$\underline{\mathbb{P}}(\mathcal{M}) = \textstyle\sum_{\theta\in\Theta:\Gamma(\theta)\subseteq\mathcal{M}} \mathbb{P}(\theta)\,, \quad \overline{\mathbb{P}}(\mathcal{M}) = \textstyle\sum_{\theta\in\Theta:\Gamma(\theta)\cap\mathcal{M}\neq\emptyset} \mathbb{P}(\theta)\,. \tag{5}$$

Suppose we are interested in the probability of $\mathbf{Q}$, an event assigning truth values to ground atoms in the Herbrand base of the union of program $\mathbf{P}$ with all facts in $\mathbf{PF}$. A direct translation of Expression (5) leads to an algorithm that computes bounds on $\mathbb{P}(\mathbf{Q})$ as follows:

- **Given** a PLP $\langle \mathbf{P}, \mathbf{PF}\rangle$ and $\mathbf{Q}$, initialize a and b with 0.
- For each total choice θ of probabilistic facts, compute the set $\Gamma(\theta)$ of all stable models of $\langle \mathbf{P}, \theta\rangle$, and:
  - if every stable model in $\Gamma(\theta)$ satisfies the truth values assigned by $\mathbf{Q}$, then add $\mathbb{P}(\theta)$ to a;
  - if some stable model in $\Gamma(\theta)$ satisfies the truth values assigned by $\mathbf{Q}$, then add $\mathbb{P}(\theta)$ to b.
- **Return** $[a, b]$ as the interval $[\underline{\mathbb{P}}(\mathbf{Q})\,, \overline{\mathbb{P}}(\mathbf{Q})]$.

Note that to find whether every stable model of a program satisfies the truth values assigned by $\mathbf{Q}$, we must run *cautious inference*, and to find whether some stable model of a program satisfies the truth values assigned by $\mathbf{Q}$, we must run *brave inference*.

For infinitely monotone credal sets we can find easy expressions for lower and upper *conditional* probabilities (that is, the infimum and supremum of conditional probabilities). Indeed, if A and B are events, then the lower probability of A given B is (Augustin et al., 2014):

$$\underline{\mathbb{P}}(A|B) = \frac{\underline{\mathbb{P}}(A \cap B)}{\underline{\mathbb{P}}(A \cap B) + \overline{\mathbb{P}}(A^c \cap B)} \tag{6}$$

when $\underline{\mathbb{P}}(A \cap B) + \overline{\mathbb{P}}(A^c \cap B) > 0$; we also have that $\underline{\mathbb{P}}(A|B) = 1$ when $\underline{\mathbb{P}}(A \cap B) + \overline{\mathbb{P}}(A^c \cap B) = 0$ and $\overline{\mathbb{P}}(A \cap B) > 0$; finally, $\underline{\mathbb{P}}(A|B)$ is undefined when $\overline{\mathbb{P}}(A \cap B) = \overline{\mathbb{P}}(A^c \cap B) = 0$ (as this condition is equivalent to $\overline{\mathbb{P}}(B) = 0$). Similarly, the upper probability of A given B is:

$$\overline{\mathbb{P}}(A|B) = \frac{\overline{\mathbb{P}}(A \cap B)}{\overline{\mathbb{P}}(A \cap B) + \underline{\mathbb{P}}(A^c \cap B)} \tag{7}$$

when $\overline{\mathbb{P}}(A \cap B) + \underline{\mathbb{P}}(A^c \cap B) > 0$; and we have that $\overline{\mathbb{P}}(A|B) = 0$ when $\overline{\mathbb{P}}(A \cap B) + \underline{\mathbb{P}}(A^c \cap B) = 0$ and $\overline{\mathbb{P}}(A^c \cap B) > 0$; finally, $\overline{\mathbb{P}}(A|B)$ is undefined when $\overline{\mathbb{P}}(A \cap B) = \overline{\mathbb{P}}(A^c \cap B) = 0$. We also note that the computation of lower and upper *expected values* with respect to infinitely monotone Choquet capacities admits relatively simple expressions (Wasserman & Kadane, 1992). For instance, the *lower expectation* $\underline{\mathbb{E}}[f] = \inf_{\mathbb{P} \in \mathbb{K}} \mathbb{E}_{\mathbb{P}}[f]$, where f is a function over the truth assignments, and $\mathbb{E}_{\mathbb{P}}[f]$ is the expectation of f with respect to $\mathbb{P}$, is $\underline{\mathbb{E}}[f] = \sum_{\theta \in \Theta} \mathbb{P}(\theta) \min_{\omega \in \Gamma(\theta)} f(\omega)$ (Wasserman & Kadane, 1992). And there are expressions even for lower and upper *conditional expectations* that mirror Expression (5).

To translate these expresions into actual computations, suppose we have $\mathbf{Q}$ and $\mathbf{E}$ assigning truth values for ground atoms in the Herbrand base of the union of program $\mathbf{P}$ with all facts in $\mathbf{PF}$. To obtain bounds on $\mathbb{P}(\mathbf{Q}|\mathbf{E})$, we can combine the previous algorithm with Expressions (6) and (7), to obtain:

- **Given** a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ and $\mathbf{Q}$, initialize a, b, c, and d with 0.
- For each total choice θ of probabilistic facts, compute the set $\Gamma(\theta)$ of all stable models of $\langle \mathbf{P}, \theta \rangle$, and:
  - if every stable model in $\Gamma(\theta)$ satisfies all truth values assigned by $\mathbf{Q}$ and by $\mathbf{E}$, then add $\mathbb{P}(\theta)$ to a;
  - if some stable model in $\Gamma(\theta)$ satisfies all truth values assigned by $\mathbf{Q}$ and by $\mathbf{E}$, then add $\mathbb{P}(\theta)$ to b;
  - if every stable model in $\Gamma(\theta)$ satisfies all truth values assigned by $\mathbf{E}$ and fails to satisfy some truth value assigned by $\mathbf{Q}$, then add $\mathbb{P}(\theta)$ to c;
  - if some stable model in $\Gamma(\theta)$, satisfies all truth values assigned by $\mathbf{E}$ and fails to satisfy some truth value assigned by $\mathbf{Q}$, then add $\mathbb{P}(\theta)$ to d.
- **Return** the interval $[\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}), \overline{\mathbb{P}}(\mathbf{Q}, \mathbf{E})]$ as follows, in case $b + d > 0$ (otherwise, report failure and stop):
  - $[0, 0]$ if $b + c = 0$ and $d > 0$;
  - $[1, 1]$ if $a + d = 0$ and $b > 0$;
  - $[a/(a + d), b/(b + c)]$ otherwise.

In fact the algorithm above was derived by Calì, Lukasiewicz, Predoiu, and Stuckenschmidt (2009), using clever optimization techniques (however they use a different strategy to handle the case $\overline{\mathbb{P}}(\mathbf{E}) = 0$). The advantage of our approach is that the algorithm is a transparent consequence of known facts about capacities; other than that, Cali et al. have already presented the algorithm so we do not need to dwell on it. Rather, we later use this algorithm to show results on the complexity of computing the lower probability $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E})$.

Algorithms that reproduce some properties of infinitely monotone Choquet capacities are also presented by Michels et al. (2015) in their work on constraint logic programming.

5. The Complexity of Inferences: Acyclic and Stratified Probabilistic Logic Programs

In this section we focus on the computation of inferences for acyclic and stratified PLPs; in these cases both the credal and the well-founded semantics agree. Section 5.1 deals with acyclic PLPs, and Section 5.2 concentrates on stratified ones.

Whenever results take as input a PLP, we assume that the size of the input consists of the length of the bitstrings encoding the program $\mathbf{P}$ and the probabilistic facts $\mathbf{PF}$ (for instance, using a textual representation). Similarly, additional input (such as sets $\mathbf{Q}$ and $\mathbf{E}$ in the next definition) are given as bitstrings and rational numbers are encoded as pairs of integers in binary notation.

We focus on the following decision problem:

Input: A PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ whose probabilities are rational numbers, a pair $(\mathbf{Q}, \mathbf{E})$, called the *query*, where both $\mathbf{Q}$ and $\mathbf{E}$ assign truth values to ground atoms in the Herbrand base of the union of program $\mathbf{P}$ and all facts in $\mathbf{PF}$, and a rational $\gamma \in [0, 1]$.

Output: Whether or not $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$; by convention, output is NO (that is, input is rejected) if $\mathbb{P}(\mathbf{E}) = 0$.

We refer to this complexity as the *inferential complexity* of PLPs. One may also be interested in the complexity of inferences when the PLP is fixed, and the only input is the query $(\mathbf{Q}, \mathbf{E})$. This is the *query complexity* of the program; to define it, consider:

Fixed: A PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$, whose probabilities are rational numbers, that employs a vocabulary $\mathbf{R}$ of predicates.

Input: A pair $(\mathbf{Q}, \mathbf{E})$, called the *query*, where both $\mathbf{Q}$ and $\mathbf{E}$ assign truth values to ground atoms of predicates in $\mathbf{R}$, and a rational $\gamma \in [0, 1]$.

Output: Whether or not $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$; by convention, output is NO if $\mathbb{P}(\mathbf{E}) = 0$.

In this case, the size of the input is the length of the bitstring specifying the sets $\mathbf{Q}$ and $\mathbf{E}$ (for instance, in textual representation) and the rational γ (represented as a pair of integers in binary notation).

Say that the query complexity of a class $\mathcal{P}$ of PLPs is in a complexity class C if the complexity of this decision problem is in C for every PLP in $\mathcal{P}$. And say that the query complexity of $\mathcal{P}$ is C-hard if each decision problem in C can be reduced, with many-one reductions, to a decision problem for at least one PLP in $\mathcal{P}$. And say that the query complexity of $\mathcal{P}$ is C-complete if $\mathcal{P}$ is both in C and C-hard.

In practice, one may face situations where a PLP may be small compared to the query, or where a single PLP is queried many times; then query complexity is the concept of interest.

The definition of query complexity is clearly related to the concept of *data complexity* found in database theory (Abiteboul, Hull, & Vianu, 1995); indeed we have used "data complexity" in previous related work (Cozman & Mauá, 2015a). Here we prefer to use

"query" instead of "data" because usually data complexity fixes the rules and varies the number of facts, while we keep both rules and facts fixed. In fact, a study of complexity where rules and query are fixed and facts vary is provided by Ceylan et al. (2016); they appropriately refer to "data" complexity, as their queries are unions of Boolean conjunctive formulas as employed in databases (Date, 2005).

A few parallel results by Ceylan et al. (2016) deserve discusion. They analyze the complexity of PLPs under two semantics; one of them is in line with Sato's distribution semantics, and another one is geared towards inconsistent programs, but neither is equivalent to the credal or the well-founded semantics (they do not use undefined values, and they only consider answers that are true in *all* models of a program). Moreover, they do not allow for conditioning evidence, and they use a somewhat different version of probabilistic facts called *contexts* (that can be reproduced with our probabilistic facts). Finally, and as noted already, they focus on unions of Boolean conjunctive queries; this leads them to complexity results that are distinct from ours. Despite these differences, they prove statements that are related to results in Section 5.1. More precisely: by translating various languages and arguments appropriately, points made by our Theorems 16 and 17 can be obtained from their results on *full acyclic* programs; also our Theorem 20 is comparable to their corresponding result, even though our "query" complexity is not their "data" complexity.[3] We decided to include our proof of Theorem 17 in full here because we need the techniques in later proofs, and because we find that our techniques illuminate the matter adequately.

5.1 Acyclic Probabilistic Logic Programs

We start with acyclic PLPs. In this case the credal and the well-founded semantics define a single distribution, given by a Bayesian network whose structure is the program's grounded dependency graph, and whose parameters are obtained from the program's Clark completion (Poole, 1993, 2008).

Example 14. Take a simplified version of the PLP in Example 5, without predicates calls and neighbor:

$$0.7 :: \mathsf{burglary}. \qquad 0.2 :: \mathsf{earthquake}.$$
$$\mathsf{alarm} :- \mathsf{burglary}, \mathsf{earthquake}, \mathsf{a1}.$$
$$\mathsf{alarm} :- \mathsf{burglary}, \mathbf{not}\ \mathsf{earthquake}, \mathsf{a2}.$$
$$\mathsf{alarm} :- \mathbf{not}\ \mathsf{burglary}, \mathsf{earthquake}, \mathsf{a3}.$$
$$0.9 :: \mathsf{a1}. \quad 0.8 :: \mathsf{a2}. \quad 0.1 :: \mathsf{a3}.$$

We can understand this PLP as the specification of the Bayesian network in Figure 3. Note that the structure of the network is just the grounded dependency graph, and the logical sentence comes directly from the Clark completion. □

Conversely, any propositional Bayesian network can be specified by an acyclic propositional PLP (Poole, 1993, 2008). The argument is simple, and we show it by turning Example 14 upside down:

3. We note that our results on acyclic programs appeared (Cozman & Mauá, 2016) almost simultaneously to the publication by Ceylan et al. (2016), and were produced independently.

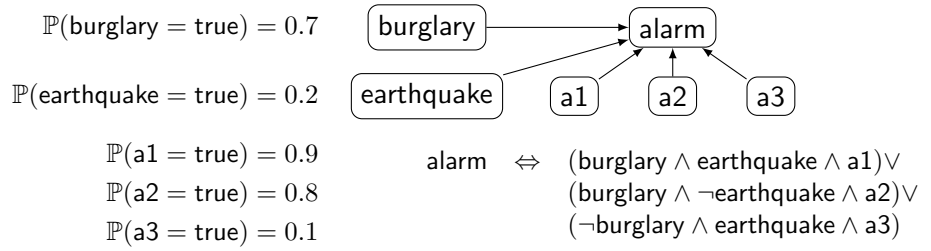

Figure 3: Bayesian network extracted from the propositional portion of Example 5.

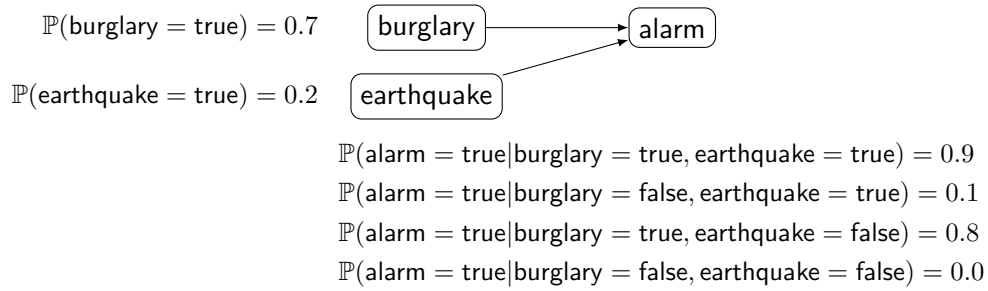

Figure 4: Bayesian network equivalent to the Bayesian network in Figure 3.

Example 15. Suppose we have the Bayesian network in Figure 4. This Bayesian network is equivalent to the Bayesian network in Figure 3 (that is: the same distribution is defined over alarm, burglary, earthquake). And the latter network is specified by an acyclic PLP. □

By combining these arguments, we should suspect that inference in acyclic propositional PLPs has the complexity of inference in Bayesian networks (Darwiche, 2009; Roth, 1996). Indeed, we have (we prove the following result in detail so that we can later use the arguments for membership):

Theorem 16. *The inferential complexity of acyclic propositional* PLP*s is* PP*-complete.*

Proof. To prove membership, start with the "unconditional" decision $\mathbb{P}(\mathbf{Q}) > 1/2$. Consider a nondeterministic polynomial-time Turing machine that starts by building a total choice as follows. The machine goes over the probabilistic facts, one by one. Take a probabilistic fact $\alpha_i :: A_i\mathbf{.}$, and suppose first that $\alpha_i = 1/2$. Then the machine simply decides, nondeterministically, whether A_i is in the total choice or not, and moves to the next fact. Now consider the general case where α_i is a rational such that $\alpha_i = \mu_i/\nu_i$ for some (smallest) integers μ_i and ν_i. Then the machine nondeterministically selects one amongst ν_i possible transitions: for μ_i of these transitions, A_i is in the total choice, and for $\nu_i - \mu_i$ transitions, A_i is not in the total choice. Then, for the generated total choice, the machine processes the resulting acyclic normal logic program: logical reasoning can determine whether $\mathbf{Q}$ is satisfied or not, by determining whether some atoms are true or false with acyclic propositional logic programs requires polynomial effort (Eiter et al., 2007, Table 2). To understand that this procedure solves our problem, reason as follows. Suppose we have n probabilistic facts; there is a total of $\prod_i \nu_i$ computation paths, and a particular total choice θ is produced by $\prod_{i:A_i\in\theta} \mu_i \prod_{j:A_i\notin\theta}(\nu_j - \mu_j)$ computation paths that lead to acceptance if and only if θ

induces the truth values assigned by $\mathbf{Q}$. Denote by $\mathbb{I}_{\mathbf{Q}}(\theta)$ the function that yields 1 if θ induces the truth values assigned by $\mathbf{Q}$, and 0 otherwise. Thus the number of accepting computations divided by the total number of computations is

$$\frac{\sum_\theta \mathbb{I}_{\mathbf{Q}}(\theta) \prod_{i:A_i\in\theta} \mu_i \prod_{j:A_i\notin\theta}(\nu_j - \mu_j)}{\prod_i \nu_i} = \sum_\theta \mathbb{I}_{\mathbf{Q}}(\theta) \prod_{i:A_i\in\theta} \alpha_i \prod_{j:A_i\notin\theta} (1-\alpha_j),$$

and the latter summation is equal to $\mathbb{P}(\mathbf{Q})$, so we obtain that $\mathbb{P}(\mathbf{Q}) > 1/2$ if and only if more than half of the computations accept for the nondeterministic polynomial-time Turing machine. Hence the decision problem is in PP.

Now consider membership of the decision $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$; we process this decision as follows.[4] Suppose the query consists of $\mathbf{Q} = \{Q_1, \ldots, Q_n\}$ and $\mathbf{E} = \{E_1, \ldots, E_m\}$, where each Q_i and each E_j is the assignment of a truth value to a particular ground atom. The simplest case is $\gamma \geq 1/2$, so assume it to begin. Then, as the query is processed, introduce $\mathsf{aux1} :- Q_1, \ldots, Q_n.$, $\mathsf{aux2} :- E_1, \ldots, E_m.$, $\mathsf{aux3} :- \mathsf{aux1}, \mathsf{aux2}, \mathsf{aux4}.$, $\mathsf{aux3} :-$ **not** $\mathsf{aux2}, \mathsf{aux5}.$, where each Q_i or E_j is written as the corresponding subgoal, and $(1/(2\gamma))$ :: $\mathsf{aux4}.$, 0.5 :: $\mathsf{aux5}.$. Thus $\mathbb{P}(\mathsf{aux3} = \mathsf{true}) > 1/2 \Leftrightarrow (1/(2\gamma))\mathbb{P}(\mathbf{Q} \cap \mathbf{E}) + (1/2)(1 - \mathbb{P}(\mathbf{E})) > 1/2 \Leftrightarrow \mathbb{P}(\mathbf{Q} \cap \mathbf{E}) > \gamma$ (that is, the decision on $\mathbb{P}(\mathsf{aux3} = \mathsf{true}) > 1/2$ yields the decision on $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$). Now if $\gamma < 1/2$, then use the same rules for aux1 and aux2, and introduce $\mathsf{aux3} :- \mathsf{aux1}, \mathsf{aux2}.$, $\mathsf{aux3} :-$ **not** $\mathsf{aux1}, \mathsf{aux2}, \mathsf{aux6}.$, $\mathsf{aux3} :-$ **not** $\mathsf{aux2}, \mathsf{aux5}.$, 0.5 :: $\mathsf{aux5}.$, and $(1 - 2\gamma)/(2 - 2\gamma)$ :: $\mathsf{aux6}.$. Thus $\mathbb{P}(\mathsf{aux3} = \mathsf{true}) > 1/2 \Leftrightarrow \mathbb{P}(\mathbf{Q} \cap \mathbf{E}) + (1 - 2\gamma)/(2 - 2\gamma)(\mathbb{P}(\mathbf{E}) - \mathbb{P}(\mathbf{Q} \cap \mathbf{E})) + (1/2)(1 - \mathbb{P}(\mathbf{E})) > 1/2 \Leftrightarrow \mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$, as desired.

Hardness is immediate as a Bayesian network over binary variables can be encoded as an acyclic PLP with polynomial effort. □

We can apply a similar reasoning to acyclic non-propositional PLPs: ground the program into a propositional PLP and then transform into a Bayesian network. If the arity of predicates is assumed bounded, then the Herbrand base contains polynomially many ground atoms. However, even when predicate arity is bounded, the grounded program, and hence the corresponding Bayesian network, can be exponentially large (as there is no bound on the number of atoms that appear in a single rule, each rule may have many logical variables, thus leading to many groundings). In fact, the complexity for non-propositional acyclic programs climbs one level further in the polynomial hierachy:

Theorem 17. *The inferential complexity of acyclic* PLP*s with bounded predicate arity is* $\mathsf{PP}^{\mathsf{NP}}$*-complete.*

Proof. To prove membership, just follow exactly the same reasoning as in the proof of Theorem 16 (adapting to the non-propositional setting in a few places). That is, start with the "unconditional" decision $\mathbb{P}(\mathbf{Q}) > 1/2$, and note that this decision problem is in $\mathsf{PP}^{\mathsf{NP}}$ as logical reasoning with acyclic normal logic programs is P^{NP}-complete (Eiter et al., 2007, Table 5). And to decide $\mathbb{P}(\mathbf{Q}|\mathbf{E}) > \gamma$, just reproduce the computations in the second paragraph of the proof of membership for Theorem 16.

4. This technique is used several times in latter proofs; we are indebted to Cassio Polpo de Campos for suggesting it. The probabilities attached to probabilistic facts are as proposed by Park and described by Darwiche (2009, Thm. 11.5).

Hardness is shown by building a PLP that solves the problem $\#_1\mathsf{3CNF}(>)$ as defined in Section 2.3; that is, one has a propositional sentence ϕ in 3CNF with two sets of logical variables $\mathbf{X}$ and $\mathbf{Y}$, and the goal is to decide whether the number of truth assignments for $\mathbf{X}$ that satisfy $\exists \mathbf{Y} : \phi(\mathbf{X}, \mathbf{Y})$ is larger than a given integer M. We take ϕ to be a conjunction of clauses $c_1, \ldots c_k$; each clause c_j contains an ordered triplet of propositional variables.

For instance, we might have as input the integer $M = 1$ and the formula

$$\varphi(X_1, X_2, Y_1) \equiv (\neg X_1 \vee X_2 \vee Y_1) \wedge (X_1 \vee \neg X_2 \vee Y_1) \wedge (\neg Y_1 \vee \neg Y_1 \vee \neg Y_1). \tag{8}$$

In this case the input is accepted (the number of satisfying assignments is 2). Note that the last clause is equivalent to $\neg Y_1$; we pad the clause so as to have three literals in it.

To encode an input instance of $\#_1\mathsf{3CNF}(>)$, we build a PLP with a few symbols. We use a set of logical variables, one per propositional variable Y_i. To simplify the notation, we denote the logical variable associated with Y_i by the same symbol. The ordered tuple of propositional variables in clause c_j corresponds to a tuple of propositional variables that is denoted by $\mathbf{Y}_j$; these are the propositional variables in c_j that belong to $\mathbf{Y}$. In Expression (8), $\mathbf{Y}_1 = \mathbf{Y}_2 = \mathbf{Y}_3 = [Y_1]$. We also use two constants, $\mathsf{0}$ and $\mathsf{1}$, standing for false and true respectively. Also, we use 0-arity predicates $\mathsf{x_i}$, each one standing for a propositional variable X_i in $\mathbf{X}$. And we use predicates $\mathsf{c_1}, \ldots, \mathsf{c_k}$, each one standing for a clause c_j. The arity of each $\mathsf{c_j}$ is the length of $\mathbf{Y}_j$, denoted by d_j. Finally, we use a 0-arity predicate cnf.

The probabilistic facts of our program are (one per predicate $\mathsf{x_i}$):

$$0.5 :: \mathsf{x_i}.$$

In addition, a set of facts is introduced as follows. For each $\mathsf{c_j}$, loop through the 2^{d_j} possible assignments of $\mathbf{Y}_j$. That is, if $d_j = 1$, then go over $\mathsf{c_j(0)}$ and $\mathsf{c_j(1)}$; if $d_j = 2$, then loop through $\mathsf{c_j(0,0)}$, $\mathsf{c_j(0,1)}$, $\mathsf{c_j(1,0)}$ and $\mathsf{c_j(1,1)}$. And if $d_j = 3$, go over the 8 assignments. Note that if $d_j = 0$, there is only one "empty" assignment to visit. Thus there are at most $8k$ assignments to visit. Consider then predicate $\mathsf{c_j}$ and an assignment $\mathbf{y}$ of $\mathbf{Y}_j$ (which may be empty). If c_j is true for $\mathbf{y}$, regardless of the possible assignments for propositional variables X_i, then just introduce the fact

$$\mathsf{c_j}(\mathbf{y}).$$

If instead c_j is false for $\mathbf{y}$, regardless of the possible assignments for propositional variables X_i, then just move to another assignment (that is, there are no propositional variables X_i in c_j, and the clause is false for $\mathbf{y}$; by ommitting $\mathsf{c_j}(\mathbf{y})$, we guarantee that it is assigned to false by the semantics). Otherwise, there are propositional variables in $\mathbf{X}$ that affect the truth value of c_j when $\mathbf{y}$ is fixed; there may be one, two or three such propositional variables. Take the first one of them, denoted by X_{j1}, and introduce the rule

$$\mathsf{c_j}(\mathbf{y}) :- \begin{cases} \mathsf{x_{j1}}. & \text{if the literal for } X_{j1} \text{ does not contain negation; or} \\ \textbf{not } \mathsf{x_{j1}}. & \text{if the literal for } X_{j1} \text{ contains negation.} \end{cases}$$

If there is a second propositional variable X_{j2} that affects the truth value of c_j when $\mathbf{y}$ is fixed, add a similar rule $\mathsf{c_j}(\mathbf{y}) :- [\textbf{not}]\ \mathsf{x_{j2}}$. And similarly if there is a third propositional variable X_{j3} that affects the truth value of c_j. Note that these rules create a disjunction for $\mathsf{c_j}$, in effect encoding the clause c_j for fixed $\mathbf{y}$.

Finally, introduce the rule

$$\mathsf{cnf} :- \mathsf{c}_1(\mathbf{Y}_1), \mathsf{c}_2(\mathbf{Y}_2), \ldots, \mathsf{c}_k(\mathbf{Y}_k).$$

The Clark completion of the PLP just constructed encodes the $\#_1\mathsf{3CNF}(>)$ problem of interest, thus proving $\mathsf{PP}^{\mathsf{NP}}$-hardness: to determine whether $\exists \mathbf{Y} : \phi(\mathbf{X}, \mathbf{Y})$ has more than M satisfying assignments, decide whether $\mathbb{P}(\mathsf{cnf} = \mathsf{true}) > M/2^n$, where n is the number of propositional variables in $\mathbf{X}$.

For instance, given the formula in Expression (8), generate the following PLP:

$$\begin{array}{lll}
\mathsf{c}_1(0) :- \textbf{not } \mathsf{x}_1 . & \mathsf{c}_1(0) :- \mathsf{x}_2 . & \mathsf{c}_1(1). \\
\mathsf{c}_2(0) :- \mathsf{x}_1 . & \mathsf{c}_2(0) :- \textbf{not } \mathsf{x}_2 . & \mathsf{c}_2(1). \\
\mathsf{c}_3(0). & & \\
\mathsf{cnf} :- \mathsf{c}_1(Y_1), \mathsf{c}_2(Y_1), \mathsf{c}_3(Y_1). & & \\
0.5 :: \mathsf{x}_1. & 0.5 :: \mathsf{x}_2. &
\end{array}$$

By determining whether $\mathbb{P}(\mathsf{cnf} = \mathsf{true}) > M/2^2$, we decide whether the number of truth assignments for X_1 and X_2 such that $\exists Y_1 \varphi(X_1, X_2, Y_1)$ holds is larger than M. □

Theorem 17 suggests that acyclic PLPs capture a larger set of probabilistic languages than many probabilistic relational models that stay within PP (Cozman & Mauá, 2015b). Intuitively, this results shows that, to produce an inference for a PLP with bounded predicate arity, one must go through the truth assignments for polynomially many groundings, guessing one at a time (thus a counting nondeterministic Turing machine), and, for *each* assignment, it is then necessary to use an NP-oracle to construct the probability values.

The next step is to remove the bound on arity. We obtain:

Theorem 18. *The inferential complexity of acyclic* PLP*s is* PEXP*-complete.*

Proof. Membership follows from grounding the PLP.[5] If the PLP has n constants, then a relation of arity k produces n^k groundings. Each one of these exponentially many groundings corresponds to a node of a (necessarily acyclic) Bayesian network. To write down the conditional probabilities associated with each node of the grounded Bayesian network, take the Clark completion of the program, and ground the expressions. For each non-root node we have a first-order formula that can be written as a possibly exponentially-long quantifier-free formula. Now to determine whether $\mathbb{P}(\mathbf{Q}) > 1/2$, we can resort to inference in the exponentially large grounded Bayesian network; that is, we can use some exponential-time nondeterministic Turing machine that guesses a truth assignment for all grounded probabilistic facts, and for each such truth assignment, computes the truth assignment for any other atom by going through the possibly exponentially large non-root node completions.

To prove hardness, we encode an exponential time nondeterministic Turing machine $\mathbb{M}$ using logical formulas that are directly produced by the Clark completion of an acyclic normal logic program $\mathbf{P}$. Assume that $\mathbb{M}$ can solve some PEXP-complete problem; that is, for a PEXP-complete language $\mathcal{L}$, $\ell \in \mathcal{L}$ iff $\mathbb{M}$ halts within time 2^n with more than half

5. A short proof of membership is obtained by applying the same concise argument used in the proof of Theorem 22(c); here we present a longer but possibly more intuitive argument based in inference on Bayesian networks.

of paths accepting ℓ, where n is some polynomial on the length of ℓ. This program $\mathbf{P}$ is then coupled with a set of probabilistic facts $\mathbf{PF}$, so that an inference in the resulting PLP decides whether the number of acceptings paths of $\mathbb{M}$ is larger than half of the total number of computation paths (thus deciding the language $\mathcal{L}$).

To describe the construction of $\mathbf{P}$, we need to have a more detailed description of $\mathbb{M}$. Suppose $\mathbb{M}$ has states q, with an initial state q_0, an accepting state q_a, and a rejecting state q_r; suppose also that $\mathbb{M}$ uses an alphabet with symbols σ (in the alphabet there is a blank symbol $\sqcup$); finally suppose that $\mathbb{M}$ has a transition function δ that takes a pair (q, σ), understood as state q and symbol σ read by the machine head, and returns one of a number of triplets (q', σ', m), where q' is the next state, σ' is the symbol to be written at the tape, and m is either -1 (head goes to the left), 0 (head stays at the same position), and 1 (head goes to the right).

We make an additional assumption about $\mathbb{M}$ (we will use it later): we must assume that, once $\mathbb{M}$ reaches q_a or q_r, it stays with the same configuration (it just keeps repeating the state and the tape), so that the number of accepting paths is the same number of interpretations that reach either q_a or q_r; this assumption is harmless as $\mathbb{M}$ can always be modified to do it.

The encoding of $\mathbb{M}$ is obtained by introducing a number of predicates and a number of first-order sentences $\phi_1, \ldots \phi_v$, such that when all these sentences hold, then any interpretation for the predicates is an accepting computation. There are several ways to encode a Turing machine in first-order logic; we adopt the encoding detailed by Grädel (2007, Thm. 3.2.4). We omit the logical expressions of this encoding as they can be taken from Grädel's presentation. If we decide whether the number of interpretations for the predicates in these sentences is larger than half of the number of possible interpretations, we obtain the desired decision.

We force each sentence ϕ_i to hold by introducing a predicate $\mathsf{aux_i}$ and a rule $\mathsf{aux_i} :- \phi_i$ (where we write ϕ_i in the rule with the understanding that ϕ_i is obtained as the Clark completion of a set of auxiliary predicates and rules; recall that conjunction, disjunction and negation are available, as well as existential quantifiers; universal quantifiers are produced by negating existential ones); then the sentence ϕ_i holds iff $\{\mathsf{aux_i} = \mathsf{true}\}$ holds. Denote by $\mathbf{E}$ the set of assignments $\{\mathsf{aux_i} = \mathsf{true}\}$ for all sentences $\phi_1, \ldots, \phi_v$. So, whenever $\mathbf{E}$ holds, the only possible interpretations for the predicates in Grädel's construction are the interpretations that correspond to computations of $\mathbb{M}$.

Now, we must have one of the sentences in $\mathbb{M}$'s encoding a "detector" for the accepting state; that is, $\exists \mathbf{X} : \mathsf{state}_{q_a}(\mathbf{X})$, where $\mathbf{X}$ is a set of logical variables that indexes the computation steps, and $\mathsf{state}_{q_a}(\mathbf{X})$ is a predicate that indicates that at computation step $\mathbf{X}$ the state is state_{q_a} (such a predicate is available in Grädel's construction). Introduce predicate $\mathsf{aux_a}$ and rule $\mathsf{aux_a} :- \mathsf{state}_{q_a}(\mathbf{X})$., so that we can actually detect whether q_a is reached by examining $\mathsf{aux_a}$.

At this point we can reproduce the behavior of $\mathbb{M}$ if we focus on interpretations that satisfy $\mathbf{E}$. The next step is to encode the input. This can be done by using Grädel's predicates that indicate which symbol is in each initial position of the tape; the input is then inserted by facts about those predicates.

And the final step is to count the accepting computations. Recall that once $\mathbb{M}$ reaches q_a, it stays with the same configuration; hence the number of accepting paths is the same number of interpretations that satisfy $\{\mathsf{aux_a} = \mathsf{true}\}$. Then we add, for each predicate r

that is introduced in Grädel's construction, except the ones in $\mathbf{E}$, the probabilistic fact $0.5 :: \mathsf{r}(X_1, \ldots, X_k)$, where k is the arity of r. Given all of this, the decision $\mathbb{P}(\mathsf{aux_a} = \mathsf{true}|\mathbf{E}) > 1/2$ determines whether the number of "accepting interpretations" for $\mathbb{M}$ is larger than half the number of "intepretations" for $\mathbb{M}$. Thus hardness obtains. □

Consider query complexity. The following result is handy:

Theorem 19. *Query complexity is* PP*-hard for the following* PLP*:*

$$0.5 :: \mathsf{t}(X)\,. \quad 0.5 :: \mathsf{pos}(X,Y)\,. \quad 0.5 :: \mathsf{neg}(X,Y)\,.$$

$$\mathsf{c}(Y) :\!- \mathsf{pos}(X,Y), \mathsf{t}(X)\,. \quad \mathsf{c}(Y) :\!- \mathsf{neg}(X,Y), \mathbf{not}\ \mathsf{t}(X)\,.$$

Proof. Consider a CNF formula $\varphi(X_1, \ldots, X_n)$ with clauses $c_1, \ldots, c_m$ and propositional variables $X_1, \ldots, X_n$, and an integer M. As noted in Section 2.3, deciding whether the number of truth assignments (for all propositional variables) that satisfy the formula is larger than M is a PP-complete problem. Let P_j (resp., N_j) be a vector denoting the indices of the positive (negative) literals X_i (resp., $\neg X_i$) in clause j. We can encode the truth-value of a clause c_j as $\mathsf{c}(j)$, the truth-value of X_i as $\mathsf{t}(i)$, and the occurrence of a positive (negative) literal X_i such that $i \in P_j$ (resp., $i \in N_j$) as $\mathsf{pos}(i,j)$ (resp., $\mathsf{neg}(i,j)$). So assemble a query $\mathbf{Q}$ as follows. For each $j = 1, \ldots, m$, add to the query the assignment $\{\mathsf{c}(j) = \mathsf{true}\}$. And for each $j = 1, \ldots, m$, for $i \in P_j$, add to $\mathbf{Q}$ the assignment $\{\mathsf{pos}(i,j) = \mathsf{true}\}$. And finally for each $j = 1 \ldots, m$, for $i \in N_j$, add to $\mathbf{Q}$ the assignment $\{\mathsf{neg}(i,j) = \mathsf{true}\}$.

Now if a grounding of pos or neg is not already assigned true, then assign it to false and add this assignment to $\mathbf{Q}$. The Clark completion defines $\mathsf{c}(j) \Leftrightarrow \bigvee_{i \in P_j} \mathsf{t}(i) \vee \bigvee_{i \in N_j} \neg \mathsf{t}(i)$ for every c_j. And the number of assignments to $X_1, \ldots, X_n$ that satisfy φ is larger than M iff $\mathbb{P}(\mathbf{Q}) > M/2^{2s^2+s}$ where $s = \max(m, n)$. □

Consequently:

Theorem 20. *The query complexity of acyclic* PLP*s is* PP*-complete.*

Proof. Hardness follows from Theorem 19. Membership is obtained using the same reasoning as in the proof of Theorem 16, only noting that, once the probabilistic facts are selected, logical reasoning with the resulting acyclic normal logic program can be done with polynomial effort (Dantsin et al., 2001, Thm. 5.1); thus $\mathbb{P}(\mathbf{Q}) > \gamma$ can be decided within PP. □

There are subclasses of acyclic PLPs that characterize well-known tractable Bayesian networks. An obvious one is the class of propositional acyclic programs whose grounded dependency graph has bounded treewidth, as Bayesian networks subject to such a constraint are tractable (Koller & Friedman, 2009). As another interesting example, consider the two-level networks that are processed by the *Quick-Score* algorithm (Heckerman, 1990); that is, two-level networks where the top level consists of marginally independent "diseases" and the bottom level consists of "findings" that are conditionally independent given the diseases, and that are determined by *noisy-or* gates. Such a network can be easily encoded using a propositional acyclic PLP; these PLPs inherit the fact that inference is polynomial when $\mathbf{Q}$ contains only negated atoms (that is, only false). Alas, this tractability result is quite

fragile, as "positive" evidence breaks polynomial behavior as long as $\mathsf{P} \neq \mathsf{NP}$ (Shimony & Domshlak, 2003). Yet another tractable class consists of acyclic definite propositional PLPs such that each atom is the head of at most one rule: inference in this class is polynomial when $\mathbf{Q}$ contains only true. This is obtained by noting that the Clark completion of these programs produces Bayesian networks that are specified using only conjunction, and then a polynomial algorithm obtains from results by Cozman and Mauá (2015a). This is also a fragile result:

Proposition 21. *Inference for the class of acyclic propositional* PLP*s such that each atom is the head of at most one rule is* PP*-complete even if (a)* $\mathbf{Q}$ *contains only* true *but the program contains* **not***; (b) the program is definite but* $\mathbf{Q}$ *contains* false*.*

Proof. Membership follows, for both (a) and (b), from Theorem 16. So, consider hardness. Any PLP in Case (b) produces, as its Clark completion, a Bayesian network that is specified using conjunctions; inference for this sort of Bayesian network is PP-complete when evidence can be "negative" (Cozman & Mauá, 2016, Thm. 2). Hardness for Case (a) then obtains easily, because one can use negation to turn "positive" evidence into "negative" evidence. □

5.2 Stratified Probabilistic Logic Programs

A stratified normal logic program has the useful property that its universally adopted semantics produces a single interpretation (and is equal to its stable and well-founded semantics). Because every total choice of a stratified PLP produces a stratified normal logic program, the credal/well-founded semantics of a stratified PLP is a unique distribution.

One might fear that in moving from acyclic to stratified programs we must pay a large penalty. This is *not* the case: the complexities remain the same as in Section 5.1.

Theorem 22. *For locally stratified* PLP*s, inferential complexity is* PEXP*-complete; it is* $\mathsf{PP}^{\mathsf{NP}}$*-complete for* PLP*s with bounded predicate arity; it is* PP*-complete for propositional* PLP*s. For locally stratified* PLP*s, query complexity is* PP*-complete.*

Proof. Hardness for propositional stratified PLPs follows from Theorem 16, as any acyclic program is also stratified. Membership is obtained using the same reasoning in the proof of Theorem 17, only noting that, once the probabilistic facts are selected, logical reasoning with the resulting stratified normal logic program can be done with polynomial effort (Eiter et al., 2007, Table 2).

For stratified programs with bounded predicate arity, hardness follows from Theorem 17. Membership is obtained using the same reasoning in the proof of Theorem 17; in fact that proof of membership applies directly to stratified programs with bounded arity.

For general stratified PLPs, hardness is argued as in the proof of Theorem 18. Membership follows from the fact that we can ground the PLP into an exponentially large propositional PLP. Once the (exponentially-many) probabilistic facts are selected, the Turing machine is left with a stratified propositional normal logic program, and logical inference is polynomial in the size of this program (that is, logical inference requires exponential effort).

Finally, hardness of query complexity follows from Theorem 19. Membership is obtained using the same reasoning in the proof of Theorem 17, only noting that, once the probabilistic facts are selected, logical reasoning with the resulting stratified normal logic program can be

done with polynomial effort as guaranteed by the analysis of data complexity of stratified normal logic programs (Dantsin et al., 2001). □

We noted, at the end of Section 5.1, that some sub-classes of acyclic programs display polynomial behavior. We now show an analogue result for a sub-class of definite (and therefore stratified, but possibly cyclic) programs with unary and binary predicates:

Proposition 23. *Inferential complexity is polynomial for queries containing only positive literals, for* PLPs *where predicates have arity at most two, each atom unifies with the head of at most one rule, and rules take one of the following forms:*

$$\alpha :: \mathsf{a}(X). \qquad \alpha :: \mathsf{a}(a). \qquad \alpha :: \mathsf{r}(X,Y).$$
$$\mathsf{a}(X) :- \mathsf{a}_1(X), \ldots, \mathsf{a}_k(X). \qquad \mathsf{a}(X) :- \mathsf{r}(X,Y). \qquad \mathsf{a}(X) :- \mathsf{r}(Y,X).$$

Proof. We show that inference can be reduced to a tractable weighted model counting problem. First, ground the program in polynomial time (because each rule has at most two logical variables). Since the resulting program is definite, only the atoms that have a directed path (in the grounded dependency graph) to some atom in the query are relevant for determining the truth-value of the query (this follows as resolution is complete for propositional definite programs). Thus, discard all atoms that are not relevant. For the query to be true, the remaining atoms that are not probabilistic facts are forced to be true by the semantics. So collect all rules of the sort $\mathsf{a}(a) :- \mathsf{r}(a,b).$, $\mathsf{a}(a) :- \mathsf{r}(b,a).$, plus all facts and all probabilistic facts. This is an acyclic program, so that its Clark completion gives the stable model semantics. This completion is a formula containing a conjunction of subformulas $\mathsf{a}(a) \Leftrightarrow \bigvee_b \mathsf{r}(a,b)$, $\mathsf{a}(a) \Leftrightarrow \bigvee_b \mathsf{r}(b,a)$, and unit (weighted) clauses corresponding to (probabilistic) facts. The query is satisfied only on models where the lefthand side of the definitions are true; computing a probability $\mathbb{P}(\mathsf{a}(a) = \mathsf{true}|\mathbf{R})$, where $\mathbf{R}$ contains atoms $\mathsf{r}(a,b)$ in the query) is equivalent to a weighted model counting of the CNF formula $\wedge_a(\bigvee_b \mathsf{r}(a,b) \wedge \bigvee_b \mathsf{r}(b,a))$. This problem has been shown to be polynomial-time solvable (Mauá & Cozman, 2015). □

Example 24. Consider a collaborative product review website where product items are reviewed by users. The assignments of products to users are performed independently and uniformly with probability 0.2. The following PLP models the assignment process:

$$0.2 :: \mathsf{review}(X,Y). \qquad \mathsf{product}(X) :- \mathsf{review}(Y,X). \qquad \mathsf{user}(X) :- \mathsf{review}(X,Y).$$

Suppose we are interested in computing the probability that every product is assigned to at least one user, and that every user is assigned to at least one product. Assume products p_1 and p_2 and users u_1, u_2, u_3. The probability that $\{\mathsf{product}(p_1) = \mathsf{true}, \mathsf{product}(p_2) = \mathsf{true}\}$, and that $\{\mathsf{user}(u_1) = \mathsf{true}, \mathsf{user}(u_2) = \mathsf{true}, \mathsf{user}(u_3) = \mathsf{true}\}$ is equivalent to a weighted model counting problem with formula

$$[\mathsf{r}(u_1,p_1) \vee \mathsf{r}(u_2,p_1) \vee \mathsf{r}(u_3,p_1)] \wedge [\mathsf{r}(u_1,p_2) \vee \mathsf{r}(u_2,p_2) \vee \mathsf{r}(u_3,p_2)] \wedge$$

$$[\mathsf{r}(u_1,p_1) \vee \mathsf{r}(u_1,p_2)] \wedge [\mathsf{r}(u_2,p_1) \vee \mathsf{r}(u_2,p_2)],$$

where the weight of a model with n atoms $\mathsf{r}(p,u)$ assigned true is $0.2^n 0.8^{6-n}$. □

6. The Complexity of Inferences: Credal Semantics

Now consider PLPs that may be non-stratified. As we have noted previously, there are several possible semantics for these PLPs. In this section we focus on the credal semantics; we leave the well-founded semantics to Section 7.

We have to adapt the definitions of inferential and query complexity to account for the fact that we now have lower and upper probabilities. First we focus on lower probabilities; the *lower-probability* version of inferential complexity for a class of PLPs is the complexity of the following decision problem:

Input: A PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ whose probabilities are rational numbers, a pair $(\mathbf{Q}, \mathbf{E})$, called the *query*, where both $\mathbf{Q}$ and $\mathbf{E}$ assign truth values to atoms in the Herbrand base of the union of program $\mathbf{P}$ and all facts in $\mathbf{PF}$, and a rational $\gamma \in [0, 1]$.

Output: Whether or not $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) > \gamma$; by convention, output is NO (that is, input is rejected) if $\overline{\mathbb{P}}(\mathbf{E}) = 0$.

The *lower-probability* version of query complexity is, accordingly:

Fixed: A PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$, whose probabilities are rational numbers, that employs a vocabulary $\mathbf{R}$ of predicates.

Input: A pair $(\mathbf{Q}, \mathbf{E})$, called the *query*, where both $\mathbf{Q}$ and $\mathbf{E}$ assign truth values to atoms of predicates in $\mathbf{R}$, and a rational $\gamma \in [0, 1]$.

Output: Whether or not $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) > \gamma$; by convention, output is NO if $\overline{\mathbb{P}}(\mathbf{E}) = 0$.

So, we are ready to state our main results on complexity for the credal semantics. To understand these results, consider the computation of lower probabilities by the algorithms in Section 4: the basic idea is to loop through all possible configurations of probabilistic facts; each configuration requires runs of cautious/brave inference (for instance, it is necessary to check whether all possible stable models satisfy $\mathbf{Q} \cap \mathbf{E}$, and whether all possible stable models fail to satisfy $\mathbf{Q}$ while satisfying $\mathbf{E}$). Thus the proof strategies employed previously can be adapted by using cautious/brave inference in our Turing machines. We have:

Theorem 25. *Adopt the credal semantics for* PLP*s, and assume that input* PLP*s are consistent. The lower-probability version of inferential complexity is* $\mathsf{PP}^{\mathsf{NP}^{\mathsf{NP}}}$*-complete for* PLP*s where all predicates have a bound on arity, and* $\mathsf{PP}^{\mathsf{NP}}$*-complete for propositional* PLP*s. The lower-probability version of query complexity is* $\mathsf{PP}^{\mathsf{NP}}$*-complete.*

Proof. We first focus on propositional programs.

To prove membership, we describe a polynomial time nondeterministic Turing machine such that more than half of its computation paths, on a given input, end up accepting iff the input is a YES instance. The machine receives the PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$, the pair $(\mathbf{Q}, \mathbf{E})$, and the rational $\gamma \in [0, 1]$ as input. In case $\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) + \overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E}) > 0$, where we use $\neg\mathbf{Q}$ to indicate the completement of the event $\mathbf{Q}$, we have to decide whether:

$$\frac{\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E})}{\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) + \overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E})} > \gamma \quad \Leftrightarrow \quad (1-\gamma)\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > \gamma\overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E})\,.$$

Write γ as μ/ν for the smallest possible integers μ and ν, with $\nu > 0$, to conclude that our decision is whether

$$(\nu - \mu)\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > \mu\overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E})\,. \tag{9}$$

In case $\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) + \overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E}) = 0$, there are a few cases to consider, as indicated by the discussion around Expression (6). First, if $\overline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) = 0$, the machine must return NO (numeric probability value is not defined); and if $\overline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > 0$, the machine must return NO if $\gamma = 1$ and YES if $\gamma < 1$. One simple way to capture all these cases is this: if $\overline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > 0$ and $\overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E}) = 0$ and $\gamma < 1$, then return YES and stop; otherwise return YES or NO according to inequality in Expression (9). Thus the machine starts by handling the special case in the previous sentence. If $\gamma < 1$, then the machine determines whether $\overline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > 0$ and $\overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E}) = 0$ using the NP oracle twice. In each case, the machine guesses a total choice and the oracle determines, using brave inference, whether there is a stable model that satisfies the event of interest. If there is no such total choice, then the upper probability is zero. So, if $\gamma < 1$ and $\overline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > 0$ and $\overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E}) = 0$, move into the accepting state; otherwise, move to some state q and continue.

From q, the machine loops through the possible selections of probabilistic facts, operating similarly to the second algorithm in Section 4. We will use the fact that cautious logical reasoning is coNP-complete and brave logical reasoning is NP-complete (Eiter et al., 2007, Table 2).

The machine proceeds from q as in the proof of Theorem 17, nondeterministically selecting whether each fact is kept or discarded. Suppose we have n ground probabilistic facts $\alpha_1 :: A_1\bullet, \ldots, \alpha_n :: A_n\bullet$. For each probabilistic fact $\alpha_i :: A_i\bullet$, where $\alpha_i = \mu_i/\nu_i$ for smallest integers μ_i and ν_i such that $\nu_i > 0$, the machine creates μ_i transitions out of the decision to keep A_i, and $\nu_i - \mu_i$ transitions out of the decision to discard A_i. Note that after guessing the status of each probabilistic fact the machine may branch in at most ν_i paths, and the total number of paths out of this sequence of decisions is $\prod_{i=1}^{n} \nu_i$. Denote this latter number by N. At this point the machine has a normal logic program, and it runs cautious inference to determine whether $\mathbf{Q} \cap \mathbf{E}$ holds in every stable model of this program. Cautious logical reasoning is solved by the NP oracle. If indeed $\mathbf{Q} \cap \mathbf{E}$ holds in every stable model of this program, the machine moves to state q_1. Otherwise, the machine runs brave inference to determine whether $\mathbf{Q}$ is false while $\mathbf{E}$ is true in some stable model of the program. Brave logical reasoning is solved by the NP oracle. And if indeed $\mathbf{Q}$ is false while $\mathbf{E}$ is true in some stable model of the program, the machine moves to state q_2. Otherwise, the machine moves to state q_3. Denote by N_1 the number of computation paths that arrive at q_1, and similarly for N_2 and N_3. From q_1 the machine branches into $\nu - \mu$ computation paths that all arrive at the accepting state (thus there are $(\nu - \mu)N_1$ paths through q_1 to the accepting state). And from q_2 the machine branches into μ computation paths that all arrive at the rejecting state. Finally, from q_3 the machine nondeterministically moves either into the accepting or the rejecting state. Thus the number of accepting computation paths is larger than the number of rejecting computation paths iff

$$(\nu - \mu)N_1 + N_3 > \mu N_2 + N_3 \quad \Leftrightarrow \quad (\nu - \mu)\frac{N_1}{N} > \mu\frac{N_2}{N}.$$

Note that, by construction, $N_1/N = \underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E})$ and $N_2/N = \overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E})$; thus the number of accepting computation paths is larger than the number of rejecting computation paths iff

$$(\nu - \mu)\underline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > \mu\overline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E})\,.$$

Membership is thus proved.

Hardness is shown by a reduction from the problem $\#_1\mathsf{DNF}(>)$, defined to be the problem of deciding whether the number of assignments of $\mathbf{X}$ such that the formula $\phi(\mathbf{X}) = \forall\mathbf{Y} : \varphi(\mathbf{X}, \mathbf{Y})$ holds is strictly larger than M, where φ is a propositional formula in DNF with conjuncts $d_1, \ldots, d_k$ (and $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_m\}$ are sets of propositional variables). Introduce $\mathsf{x_i}$ for each X_i and $\mathsf{y_i}$ for each Y_i, and encode ϕ as follows. Each conjunct d_j is represented by a predicate $\mathsf{d_j}$ and a rule $\mathsf{d_j} :- s_1, \ldots, s_r.$, where s_i stands for a properly encoded subgoal: either some $\mathsf{x_i}$, or **not** $\mathsf{x_i}$, or some $\mathsf{y_i}$, or **not** $\mathsf{y_i}$. And then introduce k rules $\mathsf{dnf} :- \mathsf{d_j}.$, one per conjunct. Note that for a fixed truth assignment for all $\mathsf{x_i}$ and all $\mathsf{y_i}$, dnf is true iff φ holds. Now introduce probabilistic facts $0.5 :: \mathsf{x_i}$, one for each $\mathsf{x_i}$. There are then 2^n possible ways to select probabilistic facts. The remaining problem is to encode the universal quantifier over the Y_i. To do so, introduce a pair of rules for each $\mathsf{y_i}$,

$$\mathsf{y_i} :- \textbf{not } \mathsf{ny_i}. \quad \text{and} \quad \mathsf{ny_i} :- \textbf{not } \mathsf{y_i}.$$

Thus there are 2^m stable models running through assignments of $Y_1, \ldots, Y_m$, for each fixed selection of probabilistic facts. By Expression (5) we have that $\underline{\mathbb{P}}(\mathsf{dnf} = \mathsf{true})$ is equal to $\sum_\theta \min f(\omega)/2^n$, where θ denotes a total choice, the minimum is over all stable models $\omega \in \Gamma(\theta)$ produced by $\langle \mathbf{P}, \theta \rangle$, and $f(\omega)$ is a function that yields 1 if dnf is true in ω, and 0 otherwise. Now $\min f(\omega)$ yields 1 iff for all $\mathbf{Y}$ we have that $\varphi(\mathbf{X}, \mathbf{Y})$ is true, where $\mathbf{X}$ is fixed by θ. Hence $\underline{\mathbb{P}}(\mathsf{dnf} = \mathsf{true}) > M/2^n$ iff the input problem is accepted. Hardness is thus proved.

Now consider PLPs where predicates have bounded arity.

Membership follows using the same construction described for the propositional case, but using a Σ_2^P oracle as cautious logical reasoning is Π_2^P-complete and brave logical reasoning is Σ_2^P-complete (Eiter et al., 2007, Table 5).

Hardness is shown by a reduction from $\#_2\mathsf{3CNF}(>)$: Decide whether the number of assignments of $\mathbf{X}$ such that the formula $\phi(\mathbf{X}) = \forall\mathbf{Z} : \exists\mathbf{Y} : \varphi(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ holds is strictly larger than M, where φ is a propositional formula in 3CNF with clauses $c_1, \ldots, c_k$ (and $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are sets of propositional variables, where $\mathbf{X}$ contains n propositional variables). We proceed *exactly* as in the proof of hardness for Theorem 17; each propositional variable Y_i now appears as a logical variable Y_i, while each propositional variable X_i appears as a predicate $\mathsf{x_i}$. The novelty is that each propositional variable Z_i appears as a predicate $\mathsf{z_i}$; these predicates receive the same treatment as predicates $\mathsf{y_i}$ in the proof for the propositional case. So, just repeat the whole translation of the formula φ used in the proof of Theorem 17, with the only difference that now there may be propositional variables Z_i in the formula, and these propositional variables appear as predicates $\mathsf{z_i}$ in the PLP. Then introduce, for each Z_i, a pair of rules

$$\mathsf{z_i} :- \textbf{not } \mathsf{nz_i}. \quad \text{and} \quad \mathsf{nz_i} :- \textbf{not } \mathsf{z_i}.$$

Again, for each fixed selection of probabilistic facts, there are stable models, one per assignment of $\mathbf{Z}$. And $\underline{\mathbb{P}}(\mathsf{cnf} = \mathsf{true}) > M/2^n$ iff the input problem is accepted.

Finally, consider query complexity.

Membership follows using the same construction described for the propositional case, noting that cautious logical reasoning is coNP-complete and brave logical reasoning is NP-complete (Dantsin et al., 2001, Thm. 5.8).

Hardness follows again by a reduction from $\#_1\mathsf{DNF}(>)$; that is, again we must decide whether the number of assignments of $\mathbf{X}$ such that $\forall\mathbf{Y} : \varphi(\mathbf{X},\mathbf{Y})$ holds is strictly larger than M, where φ is a formula in DNF (again, the number of propositional variables in $\mathbf{X}$ is n). We employ a construction inspired by the proof of Theorem 19, using the following fixed PLP. Note that x stands for the propositional variables in $\mathbf{X}$, on which counting operates; y stands for the propositional variables in $\mathbf{Y}$, on which the universal quantifier operates; c stands for clauses that are then negated to obtain the DNF:

$$0.5 :: \mathsf{x}(V)\,.$$
$$0.5 :: \mathsf{posX}(U,V). \quad 0.5 :: \mathsf{negX}(U,V). \quad \mathsf{y}(V) :- \mathbf{not}\ \mathsf{ny}(V).$$
$$0.5 :: \mathsf{posY}(U,V). \quad 0.5 :: \mathsf{negY}(U,V). \quad \mathsf{ny}(V) :- \mathbf{not}\ \mathsf{y}(V).$$
$$\mathsf{c}(V) :- \mathsf{posX}(U,V), \mathbf{not}\ \mathsf{x}(U), \mathsf{var}(U). \quad \mathsf{c}(V) :- \mathsf{negX}(U,V), \mathsf{x}(U), \mathsf{var}(U).$$
$$\mathsf{c}(V) :- \mathsf{posY}(U,V), \mathbf{not}\ \mathsf{y}(U). \quad \mathsf{c}(V) :- \mathsf{negY}(U,V), \mathsf{y}(U).$$
$$\mathsf{dnf} :- \mathsf{clause}(V), \mathbf{not}\ \mathsf{c}(V).$$

The atoms posX (resp., posY) indicate the occurrence of a positive literal X (resp., Y) in a conjunct, while negX (resp., negY) indicate the occurrence of a negative literal; clause denotes whether a constant represents a clause, whereas var represents a propositional variable X (var is used so that the number of ground atoms $\mathsf{x}(V)$ on which dnf depends is exactly n). Thus, by providing posX, negX, posY, negY, var and clause as $\mathbf{E}$, we can encode the formula φ. It follows that $\underline{\mathbb{P}}(\mathsf{dnf} = 1|\mathbf{E}) > M/2^n$ iff the input problem is accepted. □

Theorem 25 focuses on the computation of lower probabilities. We can of course define the *upper-probability* versions of inferential and query complexities, by replacing the decision $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) > \gamma$ with $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) > \gamma$. If anything, this latter decision leads to easier proofs of membership, for all special cases are dealt with by deciding whether

$$(\nu - \mu)\overline{\mathbb{P}}(\mathbf{Q} \cap \mathbf{E}) > \mu\underline{\mathbb{P}}(\neg\mathbf{Q} \cap \mathbf{E})\,,$$

where again $\gamma = \mu/\nu$. All other points in the membership proofs remain the same, once brave and cautious reasoning are exchanged. Several arguments concerning hardness can also be easily adapted. For instance, $\mathsf{PP}^{\mathsf{NP}}$-hardness for propositional programs can be proved by reducting from $\#_1\mathsf{CNF}(>)$, by encoding a formula in CNF. Similarly, $\mathsf{PP}^{\mathsf{NP}}$-hardness for query complexity reduces from $\#_1\mathsf{CNF}(>)$ by using the fixed program described in the proof of Theorem 25 without the last rule (and query with assignments on groundings of c as in the proof of Theorem 19).

Theorem 25 does not discuss the complexity of PLPs, under the credal semantics, *without* a bound on arity. Without such a bound, *logical* cautious reasoning is coNEXP-complete, so we conjecture that exponentially bounded counting Turing machines will be needed here. We leave this conjecture as an open question.

Finally, our complexity results were obtained assuming that PLPs were consistent; of course, in practice one must consider the problem of checking consistency. We have:

Proposition 26. *Consistency checking is Π_2^P-complete for propositional* PLPs *and is Π_3^P-complete for* PLPs *where predicates have a bound on arity.*

Proof. Membership of consistency checking of a propositional PLP obtains by verifying whether logical consistency fails for some total choice of probabilistic facts. This can be accomplished by deciding whether all total choices satisfy logical consistency, as logical consistency checking for this language is NP-complete (Eiter et al., 2007, Table 1), and then flipping the result. An analogue reasoning leads to membership in Π_3^P for PLPs with a bound on arity, as logical consistency checking with bounded arity is Σ_2^P-complete (Eiter et al., 2007, Table 4).

Now consider hardness in the propositional case. Take a sentence ϕ equal to $\forall \mathbf{X} : \exists \mathbf{Z} : \varphi(\mathbf{X}, \mathbf{Z})$, where φ is a propositional formula in 3CNF with clauses $c_1, \ldots, c_k$, and vectors of propositional variables $\mathbf{X}$ and $\mathbf{Z}$. Deciding the satisfiability of such a formula is a Π_2^P-complete problem (Marx, 2011). So, introduce a predicate $\mathsf{x_i}$ for each X_i, associated with a probabilistic fact $0.5 :: \mathsf{x_i}.$, and a predicate $\mathsf{z_i}$ for each Z_i, associated with rules

$$\mathsf{z_i} :- \textbf{not } \mathsf{nz_i}. \quad \text{and} \quad \mathsf{nz_i} :- \textbf{not } \mathsf{z_i}.$$

Now encode the formula ϕ as follows. For each clause c_j with three literals, add the rules $\mathsf{c_j} :- \ell_1.$, $\mathsf{c_j} :- \ell_2.$, and $\mathsf{c_j} :- \ell_3.$, where each ℓ_i stands for a subgoal containing a predicate in $\mathsf{x_1}, \ldots, \mathsf{x_n}$ or in $\mathsf{z_1}, \ldots, \mathsf{z_m}$, perhaps preceded by **not**, as appropriate (mimicking a similar construction in the proof of Theorem 17). Then add a rule

$$\mathsf{cnf} :- \mathsf{c_1}, \ldots, \mathsf{c_k}.$$

to build the formula φ, and an additional rule

$$\mathsf{clash} :- \textbf{not } \mathsf{clash}, \textbf{not } \mathsf{cnf}.$$

to force cnf to be true in any stable model. The question of whether this program has a stable model for every configuration of $\mathbf{X}$ then solves the original question about satisfiability of ϕ.

Finally, consider hardness in the relational (bounded arity) case. Take a sentence ϕ equal to $\forall \mathbf{X} : \exists \mathbf{Z} : \neg \exists \mathbf{Y} : \varphi(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, where φ is a propositional formula in 3CNF; deciding the satisfiability of this formula is a Π_3^P-complete problem (Marx, 2011). Denote $\neg \exists \mathbf{Y} : \phi(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ by ϕ'. The strategy here will be to combine the constructs in the previous paragraph (propositional case) with the proof of hardness for Theorem 17. That is, introduce a predicate $\mathsf{x_i}$ for each X_i, associated with a probabilistic fact $0.5 :: \mathsf{x_i}.$, and a predicate $\mathsf{z_i}$ for each Z_i, again associated with rules

$$\mathsf{z_i} :- \textbf{not } \mathsf{nz_i}. \quad \text{and} \quad \mathsf{nz_i} :- \textbf{not } \mathsf{z_i}.$$

And then encode each clause c_j of φ by introducing a predicate $\mathsf{c_j}(\mathbf{Y}_j)$, where $\mathbf{Y}_j$ is exactly as in the proof of Theorem 17. And as in that proof, introduce

$$\mathsf{cnf} :- \mathsf{c_1}(\mathbf{Y}_1), \mathsf{c_2}(\mathbf{Y}_2), \ldots, \mathsf{c_m}(\mathbf{Y}_m).$$

and force ϕ' to be false by introducing:

$$\mathsf{clash} :- \textbf{not } \mathsf{clash}, \mathsf{cnf}..$$

The question of whether this program has a stable model for every configuration of $\mathbf{X}$ then solves the original question about satisfiability of ϕ. □

Note that while inference relies on the class PP with nondeterministic polynomial oracles, consistency checking relies on the class coNP with nondeterministic polynomial oracles. Even though one might suspect the latter to be somehow "easier" than the former, questions about proper inclusion in the counting hierarchy seem to be still mostly open.

Note also that we have left open the complexity of consistency checking for PLPs without a bound on predicate arity. This question should be addressed in future work.

7. The Complexity of Inferences: Well-Founded Semantics

In this section we investigate the complexity of *probabilistic* inference under the well-founded semantics. As before, we examine propositional and relational programs, and within the latter we look at programs with a bound on predicate arity (as noted before Theorem 17, the grounding of the program may still be exponentially large even with a bound on predicate arity).

Theorem 27. *Adopt the well-founded semantics for* PLP*s. The inferential complexity of* PLP*s is* PEXP*-complete; it is* $\mathsf{PP}^{\mathsf{NP}}$*-complete if the* PLP *has a bound on the arity of its predicates; it is* PP*-complete if the* PLP *is propositional. The query complexity of* PLP*s is* PP*-complete.*

Proof. Consider first propositional PLPs. Such a PLP can encode any Bayesian network over binary variables (Poole, 1993), so inference is PP-hard. Membership is proved by adapting the arguments in the proof of Theorem 16; whenever a total choice is selected by the nondeterministic Turing machine, logical inference (under the well-founded semantics) is run with polynomial effort in the resulting propositional normal logic program (Dantsin et al., 2001).

Consider now PLPs with logical variables. Membership follows from the same argument in the proof of Theorem 17, using the fact that inference in normal logic programs under the well-founded semantics is in EXP (Dantsin et al., 2001). Hardness follows from the fact that inferential complexity is PEXP-hard already for acyclic programs (Theorem 18).

Now consider PLPs with a bound on the arity of predicates. Membership follows from the same argument in the previous paragraphs, using the fact that (logical) inference in normal logic programs with a bound on the arity of predicates is, under the well-founded semantics, in P^{NP}, as proved in Theorem 28. Hardness for $\mathsf{PP}^{\mathsf{NP}}$ follows from the fact that inference complexity of PLPs under the stable model semantics is $\mathsf{PP}^{\mathsf{NP}}$-hard even for stratified programs, and noting that for stratified programs the stable model and the well-founded semantics agree. □

The proof of Theorem 27, in the case of PLPs with bound on predicate arity, uses the following result. Note that this is a result on logical inference; however it does not seem to be found in current literature.

Theorem 28. *Consider the class of normal logic programs with a bound on the arity of predicates, and consider the problem of deciding whether a literal is in the well-founded model of the program. This decision problem is* P^{NP}*-complete.*

Proof. Hardness follows from the hardness of logical inference with stratified programs under the stable model semantics (Eiter et al., 2007). Membership requires more work. We use the monotone operator $\mathbb{LFT}_{\mathbf{P}}(\mathbb{LFT}_{\mathbf{P}}(\mathcal{I}))$. Consider the algorithm that constructs the well-founded extension by starting with the empty interpretation and by iterating $\mathbb{LFT}_{\mathbf{P}}(\mathbb{LFT}_{\mathbf{P}}(\mathcal{I}))$. As there are only polynomially-many groundings, there are at most a polynomial number of iterations. Thus in essence we need to iterate the operator $\mathbb{LFT}_{\mathbf{P}}(\mathcal{I})$; thus, focus attention on the computation of $\mathbb{LFT}_{\mathbf{P}}(\mathcal{I})$. The latter computation consists of finding the least fixpoint of $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$. So we must focus on the effort involved in computing the least fixpoint of $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$. Again, there are at most a polynomial number of iterations of $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$ to be run. So, focus on a single iteration of $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$. Note that any interpretation $\mathcal{I}$ has polynomial size; however, we cannot explicitly generate the reduct $\mathcal{P}^{\mathcal{I}}$ as it may have exponential size. What we need to do then is, for each grounded atom A, to decide whether there is a rule whose grounding makes the atom A true in $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$. So we must make a nondeterministic choice per atom (the choice has the size of logical variables in a rule, a polynomial number). Hence by running a polynomial number of nondeterministic choices, we obtain an iteration of $\mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$; by running a polynomial number of such iterations, we obtain a single iteration of $\mathbb{LFT}_{\mathbf{P}}(\mathcal{I})$; and by running a polynomial number of such iterations, we build the well-founded model. Thus we are within P^{NP} as desired. □

Obviously, for the well-founded semantics there are no concerns about consistency: *every* normal logic program has a well-founded semantics, so *every* PLP has one and only one well-founded semantics.

8. The Semantics of the Credal and the Well-Founded Semantics

It does not seem that any comparison is available in the literature between the credal and the well-founded semantics. In this section we comment on some of the most notable conceptual differences between these semantics.

We start our analysis with the (three-valued) well-founded semantics. An attractive feature of this semantics is that it attaches a *unique* probability distribution to *every* well-formed PLP (even in cases where the credal semantics is not defined). Besides, the well-founded semantics for PLPs is conceptually simple for anyone who has *already* mastered the well-founded semantics for normal logic programs.

On the other hand, some of the weaknesses of the well-founded semantics already appear in non-probabilistic programs. For instance, the well-founded semantics does not "reason by cases" in a program such as (Van Gelder et al., 1991):

$$\mathsf{a} :\!- \textbf{ not } \mathsf{b}. \qquad \mathsf{b} :\!- \textbf{ not } \mathsf{a}. \qquad \mathsf{p} :\!- \mathsf{a}. \qquad \mathsf{p} :\!- \mathsf{b}.$$

The well-founded semantics leaves every atom undefined. However, it is apparent that p should be assigned true, for we can find two ways to understand the relation between a and b, and both ways take p to true (these two interpretations are exactly the stable models: one contains a and $\neg$b, the other contains $\neg$a and b). Even though we can to some extent treat undefined as indicating "lack of proof" (Vennekens et al., 2009), such an interpretation is not always justified.

Indeed, the interpretation of undefined is a difficult matter even within three-valued logic (Bergmann, 2008; Date, 2005; Malinowski, 2007; Rubinson, 2007). In short, it is

a	b	cold	headache	Probability
true	true	false	false	$0.34 \times 0.25 = 0.085$
true	false	true	true	$0.34 \times 0.75 = 0.255$
false	true	undefined	undefined	$0.66 \times 0.25 = 0.165$
false	false	false	true	$0.66 \times 0.75 = 0.495$

Table 1: Well-founded semantics for Example 29: total choices (assignments to a and b), induced assignments to cold and headache, and their probabilities.

difficult to determine whether undefined should be taken as simply an expression of subjective ignorance, or the indication that something really is neither true nor false (Wallace, 1993, Section 1.2.1.2). This unresolved debate on the semantics of three-valued logic is magnified when we mix it with probabilities. The challenge is that undefined values reflect a type of uncertainty, and probability is supposed to deal with uncertainty; by putting those together we may wish to invite collaboration but we may end up with plain confusion. Consider for instance an (adapted) example by Hadjichristodoulou and Warren (2012, Example IV.1):

Example 29. Take the PLP:

$$\mathsf{cold} :\!- \mathsf{headache}, \mathsf{a}. \qquad \mathsf{cold} :\!- \textbf{not } \mathsf{headache}, \textbf{not } \mathsf{a}. \qquad 0.34 :: \mathsf{a}.$$
$$\mathsf{headache} :\!- \mathsf{cold}, \mathsf{b}. \qquad \mathsf{headache} :\!- \textbf{not } \mathsf{b}. \qquad 0.25 :: \mathsf{b}.$$

There are four total choices, each inducing a normal logic program. In one case, namely $\{\neg\mathsf{a}, \mathsf{b}\}$, the resulting normal logic program has no stable model. Hence, this PLP has no credal semantics. However, it does have a well-founded semantics, shown in Table 1. □

Now, in this example, what does it mean to say that $\mathbb{P}(\mathsf{headache} = \mathsf{undefined}) = 0.165$? Even though it seems appropriate in this case to take undefined to signify "lack of proof" (Vennekens et al., 2009), this sort of interpretation may be challenging to practitioners. In fact, one might legitimately ask for the value of $\mathbb{P}(\mathsf{headache} = \mathsf{true} | \mathsf{headache} = \mathsf{undefined})$, not realizing that in the well-founded semantics this value is simply zero. Suppose we add to Example 29 the simple rule

$$\mathsf{c} :\!- \mathsf{a}, \mathsf{b}.$$

and one asks for $\mathbb{P}(\mathsf{c} = \mathsf{false} | \mathsf{cold} = \mathsf{undefined})$. While the well-founded semantics assigns value 1 to this probability, one might think that this probability is just equal to $\mathbb{P}(\mathsf{c} = \mathsf{false})$, given that nothing of substance is observed about cold.

The probabilistic Barber Paradox discussed in Example 11 describes a situation where the well-founded semantics can answer questions for some individuals, even as it fails to find definite answers for other questions. This is rather attractive, but one must ask: What exactly is the meaning of $\mathbb{P}(\mathsf{shaves}(b, b) = \mathsf{undefined}) = 0.5$? Note that, for the *logical program* described in Example 4, it makes sense to return an undefined value: we are at a logical corner. However, for the probabilistic program it may seem less satisfying to obtain a non-zero probability that some particular fact is undefined.

A difficulty here is that undefined values appear due to a variety of situations that should apparently be treated in distinct ways: (i) programs may be contradictory (as it happens in Example 29); (ii) programs may fail to have a clear meaning (as in the Barber Paradox); or (iii) programs may simply have several possible solutions (for instance, various stable models as in Example 10). In case (i), it is even surprising that one would try to assign probabilities to contradictory cases. In cases (ii) and (iii), probabilities may be contemplated, but then there is a confusing mix of probabilities and undefined values.

Now consider the credal semantics. There are two possible criticisms one may raise against it. First, a program may fail to have a credal semantics: consider the probabilistic Barber Paradox. Second, the credal semantics relies on sets of probability measures (credal sets), not on unique measures. We examine these two points in turn.

The fact that some programs may fail to have a credal semantics is an annoyance in that programs must be checked for consistency. However, as we have noted already, some programs seem to be contradictory, and in those cases one could argue that it is appropriate not to have semantics. So, one may be perfectly satisfied with failure in Example 29, for the total choice $\{\neg \mathsf{a}, \mathsf{b}\}$ in essence leads to the following clearly unsatisfiable pair of rules:

$$\mathsf{headache} :- \mathsf{cold}. \qquad \mathsf{cold} :- \mathbf{not}\ \mathsf{headache}.$$

Now consider the fact that the credal semantics relies on credal sets. Anyone expecting any inference to produce a single probability value may be puzzled, but reliance on sets of probabilities does not seem to be a flaw when examined in detail. One argument in favor of sets of probabilities is that they are legitimate representations for incomplete, imprecise or indeterminate beliefs (Augustin et al., 2014; Troffaes & De Cooman, 2014; Walley, 1991). But even if one is not willing to take credal sets as a final representation of beliefs, the credal semantics is wholly reasonable from a least commitment perspective. That is, the main question should always be: What are the best bounds on probabilities that one can safely assume, taking into account *only* the given rules, facts, and assessments? From this point of view, Examples 9 and 10 are entirely justified: the options given to the program are not decided by the given information, so one must leave them open.

All in all, we find that the credal semantics is conceptually stronger than the well-founded semantics, even though the latter is uniquely defined for every PLP.

An alternative, entirely different, way to avoid these complexities is to exclude from consideration programs that produce undefined atoms under some total choices. Riguzzi (2015) refers to PLPs that never produce undefined atoms as *sound* ones; these are the only ones he contemplates. Note that by staying with programs that always have an interpretation (never a *partial* one), we automatically restrict ourselves to programs that always have a unique stable model (Van Gelder et al., 1991, Corollary 5.6). Here the idea seems to be that undefined values, or non-unique stable sets, arise due to modeling error and should be banned, or perhaps avoided using additional assumptions based perhaps on causal or temporal reasoning (Vennekens et al., 2009). As argued by Riguzzi, "the uncertainty should be handled by the choices [that is, by the probabilistic facts] rather than by the semantics of negation".

	Propositional	$\mathsf{PLP_b}$	PLP	Query
Acyclic	PP	$\mathsf{PP}^{\mathsf{NP}}$	PEXP	PP
Stratified	PP	$\mathsf{PP}^{\mathsf{NP}}$	PEXP	PP
General, credal	$\mathsf{PP}^{\mathsf{NP}}$	$\mathsf{PP}^{\mathsf{NP}^{\mathsf{NP}}}$	?	$\mathsf{PP}^{\mathsf{NP}}$
General, well-founded	PP	$\mathsf{PP}^{\mathsf{NP}}$	PEXP	PP

Table 2: Complexity results. All entries refer to completeness with respect to many-one reductions. Columns "Propositional", "$\mathsf{PLP_b}$", and "PLP respectively refer to the inferential complexity of propositional PLPs, the inferential complexity of PLPs with a bound on predicate arity, and PLPs with no bound on predicate arity. Column "Query" refers to the query complexity of relational PLPs.

9. Conclusion

We can summarize our contributions as follows. First, we have identified the main ideas behind the credal and the well-founded semantics for PLPs based on probabilistic facts and normal logic programs. Other semantics may be studied in future work, but the credal and the well-founded ones seem to be the most important starting point. Second, we have shown that the credal semantics is intimately related to infinitely monotone Choquet capacities; precisely: the credal semantics of a consistent PLP is the largest credal set that dominates an infinitely monotone Choquet capacity. Third, we have derived the inferential and query complexity of acyclic, stratified and general PLPs both under the credal and the well-founded semantics. These results on complexity are summarized in Table 2; note that PLPs reach non-trivial classes in the counting hierarchy. It is interesting to note that acyclic PLPs with a bound on arity go beyond Bayesian networks in the complexity classes they can reach.

For normal logic programs (not probabilistic ones), the well-founded semantics is known to stay within lower complexity classes than the credal semantics (Dantsin et al., 2001); the phenomenon persists in the probabilistic case. One might take this as an argument for the well-founded semantics, on top of the fact that the well-founded semantics is defined for *any* PLP. On the other hand, one might argue that the credal semantics has the advantage of larger expressivity, as it can handle problems up in the counting hierarchy. In addition, our analysis in Section 8 favors, at least conceptually, the credal semantics, despite the fact that it may not be defined for some PLPs. It is much easier to understand the meaning of PLPs using the credal semantics than the well-founded semantics, as the latter mixes three-valued logic and probabilities in a non-trivial way. We suggest that more study is needed to isolate those programs where undefined values are justified and can be properly mixed with probabilities. Also, the well-founded semantics may be taken as an approximation of the set of possible probability models.

We could include in the analysis of PLPs a number of useful constructs that have been adopted in answer set programming (Eiter, Ianni, & Krennwalner, 2009). There, *classic negation*, such as $\neg\mathsf{wins}(X)$, is allowed on top of **not**. Also, constraints, such as $:- \phi$, are allowed to mean that ϕ is false. More substantial is the presence, in answer set programming,

of *disjunctive heads*, allowing rules such as $\mathsf{single}(X) \vee \mathsf{husband}(X) :- \mathsf{man}(X)$.. Now the point to be made is this. Suppose we have a probabilistic logic program $\langle \mathbf{P}, \mathbf{PF} \rangle$, where as before we have independent probabilistic facts, but where $\mathbf{P}$ is now a logic program with classic negation, constraints, disjuctive heads, and $\mathbf{P}$ is consistent in that it has stable models for every total choice of probabilistic facts. The proof of Theorem 12 can be reproduced in this setting, and hence *the credal semantics (the set of measures over stable models) of these probabilistic answer set programs is again an infinitely monotone credal set.* The complexity of inference with these constructs is left for future investigation.

Much more is yet to be explored concerning the complexity of PLPs. Several classes of PLPs deserve attention, such as definite, tight, strict, order-consistent programs, and programs with aggregates and other constructs. The inclusion of functions (with appropriate restrictions to ensure decidability) is another challenge. Concerning complexity theory itself, it seems that approximability should be investigated, as well as questions surrounding learnability and expressivity of PLPs.

Acknowledgements

The first author is partially supported by CNPq, grant 308433/2014-9, and received financial support from the São Paulo Research Foundation (FAPESP), grant 2016/18841-0. The second author is partially supported by CNPq, grants 303920/2016-5 and 420669/2016-7, and received financial support from the São Paulo Research Foundation (FAPESP), grant 2016/01055-1.

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases: the Logical Level*. Addison-Wesley Publishing Company Inc., Reading, Massachusetts.

Apt, K. R., & Bezem, M. (1991). Acyclic programs. *New Generation Computing*, *9*, 335–363.

Apt, K. R., Blair, H. A., & Walker, A. (1988). Towards a theory of declarative knowledge. In Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 193–216. Morgan Kaufmann.

Augustin, T., Coolen, F. P. A., de Cooman, G., & Troffaes, M. C. M. (2014). *Introduction to Imprecise Probabilities*. Wiley.

Baral, C., Gelfond, M., & Rushton, N. (2009). Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, *9*(1), 57–144.

Baral, C., & Subrahmanian, V. (1993). Dualities between alternative semantics for logic programming and nonmonotonic reasoning. *Journal of Automated Reasoning*, *10*(3), 399–420.

Bergmann, M. (2008). *An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems*. Cambridge University Press.

Berry, W. D. (1984). *Nonrecursive Causal Models*. Sage Publications.

Buhrman, H., Fortnow, L., & Thierauf, T. (1998). Nonrelativizing separations. In *Proceedings of IEEE Complexity*, pp. 8–12.

Calì, A., Lukasiewicz, T., Predoiu, L., & Stuckenschmidt, H. (2009). Tightly coupled probabilistic description logic programs for the semantic web. In *Journal on Data Semantics XII*, pp. 95–130. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ceylan, Í. Í., Lukasiewicz, T., & Peñaloza, R. (2016). Complexity results for probabilistic Datalog$^{\pm}$. In *European Conference on Artificial Intelligence*, pp. 1414–1422.

Clark, K. L. (1978). Negation as failure. In *Logic and Data Bases*, pp. 293–322. Springer.

Cozman, F. G., & Mauá, D. D. (2016). The complexity of Bayesian networks specified by propositional and relational languages. Tech. rep., https://arxiv.org/abs/1612.01120.

Cozman, F. G., & Mauá, D. D. (2015a). Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity. In *AAAI Conference on Artificial Intelligence*.

Cozman, F. G., & Mauá, D. D. (2015b). The complexity of plate probabilistic models. In *Scalable Uncertainty Management*, Vol. 9310 of *LNCS*, pp. 36–49. Springer.

Cozman, F. G., & Mauá, D. D. (2016). Probabilistic graphical models specified by probabilistic logic programs: Semantics and complexity. In *Conference on Probabilistic Graphical Models — JMLR Workshop and Conference Proceedings*, Vol. 52, pp. 110–121.

Dantsin, E., Eiter, T., & Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys*, *33*(3), 374–425.

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Date, C. J. (2005). *Database in Depth: Relational Theory for Practitioners*. O'Reilly.

De Raedt, L. (2008). *Logical and Relational Learning*. Springer.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (2010). *Probabilistic Inductive Logic Programming*. Springer.

Eiter, T., Faber, W., Fink, M., & Woltran, S. (2007). Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, *5*, 123–165.

Eiter, T., Ianni, G., & Krennwalner, T. (2009). Answer set programming: a primer. In *Reasoning Web*, pp. 40–110. Springer-Verlag.

Faber, W., Pfeifer, G., & Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, *175*, 278–298.

Fagin, R., & Halpern, J. Y. (1991). A new approach to updating belief. In Bonissone, P. P., Henrion, M., Kanal, L. N., & Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence 6*, pp. 347–374. Elsevier Science Publishers, North-Holland.

Fierens, D., Van den Broeck, G., Renkens, J., Shrerionov, D., Gutmann, B., Janssens, G., & De Raedt, L. (2014). Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, *15*(3), 358–401.

Fuhr, N. (1995). Probabilistic Datalog — a logic for powerful retrieval methods. In *Conference on Research and Development in Information Retrieval*, pp. 282–290, Seattle, Washington.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, Vol. 88, pp. 1070–1080.

Grädel, E. (2007). Finite model theory and descriptive complexity. In *Finite Model Theory and its Applications*, pp. 125–229. Springer.

Hadjichristodoulou, S., & Warren, D. S. (2012). Probabilistic logic programming with well-founded negation. In *International Symposium on Multiple-Valued Logic*, pp. 232–237.

Halpern, J. Y. (2003). *Reasoning about Uncertainty*. MIT Press, Cambridge, Massachusetts.

Hansen, P., & Jaumard, B. (1996). Probabilistic satisfiability. Tech. rep. G-96-31, Les Cahiers du GERAD, École Polytechique de Montréal.

Heckerman, D. (1990). A tractable inference algorithm for diagnosing multiple diseases. In *Conference on Uncertainty in Artificial Intelligence*, pp. 163–172.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Lee, J., & Wang, Y. (2015). A probabilistic extension of the stable model semantics. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, pp. 96–102.

Lukasiewicz, T. (2005). Probabilistic description logic programs. In *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, pp. 737–749, Barcelona, Spain. Springer.

Lukasiewicz, T. (2007). Probabilistic description logic programs. *International Journal of Approximate Reasoning*, *45*(2), 288–307.

Malinowski, G. (2007). Many-valued logic and its philosophy. In Gabbay, D. M., & Woods, J. (Eds.), *Handbook of the History of Logic - Volume 8*, pp. 13–94. Elsevier.

Marx, D. (2011). Complexity of clique coloring and related problems. *Theoretical Computer Science*, *412*, 3487–3500.

Mauá, D. D., & Cozman, F. G. (2015). DL-Lite Bayesian networks: A tractable probabilistic graphical model. In *Scalable Uncertainty Management*, Vol. 9310 of *LNCS*, pp. 50–64. Springer.

Michels, S., Hommersom, A., Lucas, P. J. F., & Velikova, M. (2015). A new probabilistic constraint logic programming language based on a generalised distribution semantics. *Artificial Intelligence Journal*, *228*, 1–44.

Molchanov, I. (2005). *Theory of Random Sets*. Springer.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Prentice Hall.

Nilsson, N. J. (1986). Probabilistic logic. *Artificial Intelligence*, *28*, 71–87.

Nodelman, U., Shelton, C. R., & Koller, D. (2002). Continuous time Bayesian networks. In *Conference on Uncertainty in Artificial Intelligence*, pp. 378–387.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley Publishing.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.

Pearl, J. (2009). *Causality: Models, Reasoning, and Inference (2nd edition)*. Cambridge University Press, Cambridge, United Kingdom.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, *64*, 81–129.

Poole, D. (2008). The Independent Choice Logic and beyond. In De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.), *Probabilistic Inductive Logic Programming*, Vol. 4911 of *Lecture Notes in Computer Science*, pp. 222–243. Springer.

Poole, D., & Crowley, M. (2013). Cyclic causal models with discrete variables: Markov chain equilibrium semantics and sample ordering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1060–1068.

Przymusinski, T. (1989). Every logic program has a natural stratification and an iterated least fixpoint model. In *ACM Symposium on Principles of Database Systems*, pp. 11–21.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*(1-2), 107–136.

Riguzzi, F. (2015). The distribution semantics is well-defined for all normal programs. In Riguzzi, F., & Vennekens, J. (Eds.), *International Workshop on Probabilistic Logic Programming*, Vol. 1413 of *CEUR Workshop Proceedings*, pp. 69–84.

Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*, *82*(1-2), 273–302.

Rubinson, C. (2007). Nulls, three-valued logic, and ambiguity in SQL: critiquing Date's critique. *ACM SIGMOD Record*, *36*(4), 13–17.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *Int. Conference on Logic Programming*, pp. 715–729.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, *15*, 391–454.

Sato, T., Kameya, Y., & Zhou, N.-F. (2005). Generative modeling with failure in PRISM. In *International Joint Conference on Artificial Intelligence*, pp. 847–852.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Shimony, S. E., & Domshlak, C. (2003). Complexity of probabilistic reasoning in directed-path singly-connected Bayes networks. *Artificial Intelligence*, *151*(1/2), 213–225.

Simon, J. (1975). On some central problems in computational complexity. Tech. rep. TR75-224, Department of Computer Science, Cornell University.

Tóran, J. (1991). Complexity classes defined by counting quantifiers. *Journal of the ACM*, *38*(3), 753–774.

Troffaes, M. C. M., & De Cooman, G. (2014). *Lower Previsions*. Wiley.

Van Gelder, A. (1993). The alternating fix point of logic programs with negation. *Journal of Computer and System Sciences*, *47*, 185–221.

Van Gelder, A., Ross, K. A., & Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery*, *38*(3), 620–650.

Vennekens, J., Denecker, M., & Bruynoogue, M. (2009). CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, *9*(3), 245–308.

Wagner, K. W. (1986). The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, *23*, 325–356.

Wallace, M. (1993). Tight, consistent, and computable completions for unrestricted logic programs. *Journal of Logic Programming*, *15*, 243–273.

Walley, P. (1991). *Statistical Reasoning with Imprecise Probabilities*. Chapman and Hall, London.

Wasserman, L., & Kadane, J. B. (1992). Computing bounds on expectations. *Journal of the American Statistical Association*, *87*(418), 516–522.