# The Well-Founded Semantics of Cyclic Probabilistic Logic Programs: Meaning and Complexity

**Fabio Gagliardi Cozman** [1], **Denis Deratani Mauá**[2]

[1]Escola Politécnica, Universidade de São Paulo

[2]Instituto de Matemática e Estatística, Universidade de São Paulo

***Abstract.*** *This paper investigates the properties of cyclic probabilistic normal logic programs, consisting of rules and a set of independent probabilistic facts, under the well-founded semantics. That is, we assume that for each fixed truth assignment for the probabilistic facts, the resulting normal logic program is interpreted through the well-founded semantics. We derive results concerning the complexity of inference, and discuss the challenging interplay between three-valued assignments and probabilities.*

## 1. Introduction

In this paper we focus on the combination of logic programs with probabilities. We focus on normal logic programs; that is, on sets of rules that may containing negation as failure, such as:

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

We assume that there is a set of *probabilistic facts*; that is, some facts, such as insomnia, are associated with probabilities. For instance,

$$\mathbb{P}(\mathsf{insomnia} = \mathsf{true}) = 0.3.$$

The probabilistic facts are assumed independent, hence there is a probability measure over all of them, and this probability measure can be extended to all atoms of the program in a variety of cases. This is the sort of probabilistic logic program originally proposed by Sato [27, 28] and by Poole [21, 22]. In fact, Sato focused on definite programs (that is, without negation), and Poole focused on acyclic programs (that is, no atom depends on itself). There have been a few attempts to extend these original efforts into programs that contain negation *and* cycles [16, 15, 29, 24]. To understand this combination, suppose that we add to Expression (1) the following rule:

$$\mathsf{work} :- \mathbf{not}\ \mathsf{sleep}. \tag{2}$$

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 false, and then the remaining two rules create a cycle: sleep depends on work and vice-versa. The question is how to interpret the whole probabilistic logic program.

One possible way to interpret cyclic probabilistic logic normal programs is to use the well-founded semantics whenever the truth assignment for the probabilistic facts is fixed [15]. This is the approach we follow here. We derive the complexity of inferences for propositional and relational programs (where an inference is the computation of the

probability for grounded atoms) in Section 4. We also examine the interplay between the three-valued assignments that are employed in well-founded semantics and the meaning of probabilities. In particular, we discuss some difficulties in interpreting conditioning and evidence in the presence of three-valued assignments (Section 5). We summarize our conclusions in Section 6.

## 2. Background: normal logic programs and the well-founded semantics

An *atom* is written as $\mathsf{r}(t_1, \ldots, t_k)$, where $\mathsf{r}$ is a *predicate* and each $t_i$ is a *term* (a *constant* or a *logical variable*). A *normal logic program* is a finite set of *rules* such as

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

where each $A_i$ is an atom. Atom $A_0$ is the *head*; the right hand side is the *body*. The body consists of a set of *subgoals* separated by commas ($A_1$ is a subgoal, $\mathbf{not}\ A_n$ is a subgoal). If there are no subgoals, the rule is called a *fact* and written simply as $A_0$.. Following Prolog's notation, a logical variable is always capitalized. Some normal logic programs do not contain $\mathbf{not}$; they are *definite*. An atom is *ground* if it does not contain any logical variable; a program without logical variables is *propositional*. The *Herbrand base* of a program is the set of all ground atoms build from constants and predicates mentioned in the program. By grounding every rule using atoms in the Herbrand base, we obtain the (propositional) *grounding* of a (possibly non-propositional) normal logic program. The *grounded dependency graph* of a normal logic program is a directed graph where each grounded predicate is a node, and where there is an edge from node $B$ to node $C$ if there is a grounded rule where $C$ appears in the head and $B$ appears in the body; if $B$ is preceded by $\mathbf{not}$ in any such rule, then the edge between $B$ and $C$ is *negative*.

Functions are not included in rules in this paper; we assume function-free (that is, relational) programs so that every Herbrand base is finite.

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 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 subgoal is true in an interpretation if it is an atom $A$ and $A$ belongs to the interpretation, or the subgoal is $\mathbf{not}\ A$ and $\neg A$ belongs to 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.

Now consider the semantics of normal logic programs. There are many proposals in the literature; however, currently there are two definitions that have received most attention: the *stable model* [14] and the *well-founded* [31] semantics. We now describe these semantics; alas, their definition is not simple.

Consider first 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 $\mathcal{I}$ if for each grounded rule whose head is $A$, we have that (i) some subgoal

$A_i$ or $\mathbf{not}\ A_i$ is false in $\mathcal{I}$, or (ii) some subgoal that is an atom $A_i$ 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 no atom in $\mathcal{U}$ that can be first shown to be true, as any such derivation depends on other atoms in $\mathcal{U}$. 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. Define: $A \in \mathbb{T}_{\mathbf{P}}(\mathcal{I})$ iff there is some grounded rule with head $A$ such that every subgoal in the body is true in $\mathcal{I}$. Also 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). 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.

Now, consider 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 $A_0 :- A_1, \ldots, A_m$ iff $A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n$ is a grounded rule from $\mathbf{P}$ 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 literal $\mathbf{not}\ A$ in their body such that $A$ is an atom that is true in $\mathcal{I}$, (iii) removing all remaining literals of the form $\mathbf{not}\ A$ from the remaining rules. An interpretation $\mathcal{I}$ is a *stable model* if $\mathcal{I}$ 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. It so happens that any well-founded model is a subset of every stable model of a normal logic program [31, 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 [3, 32, 23]. One is this, where the connection with the stable model semantics is emphasized [3]: write $\mathbb{LFT}_{\mathbf{P}}(\mathcal{I})$ to mean the least fixpoint of $\mathbb{F}_{\mathbf{P}}(\mathcal{I}) = \mathbb{T}_{\mathbf{P}^{\mathcal{I}}}$; 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 every atom $A$ that is *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.

The well-founded semantics determines the truth assignment for some atoms; for the remaining atoms, their "truth values are not determined by the program" [31, Section 1.3]. A very common interpretation of this situation is that the well-founded semantics uses three-valued logic where the values are true, false, and undefined.

It is instructive to look at some examples.

**Example 1.** First, take a program with rules $\mathsf{p} :- \mathbf{not}\ \mathsf{q}, \mathbf{not}\ \mathsf{r}$ and $\mathsf{q} :- \mathbf{not}\ \mathsf{p}$ (identical to Expressions (1) and (2)). The well-founded semantics assigns false to r and leaves p and q as undefined. □

**Example 2.** Consider a game where a player wins if there is another player

with no more moves [31, 32], 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: $\mathsf{move}(a,b).\ \ \mathsf{move}(b,a).\ \ \mathsf{move}(b,c).\ \ \mathsf{move}(c,d).$. Then the well-founded semantics leads to $\{\mathsf{wins}(c), \neg\mathsf{wins}(d)\}$, 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 3.** The Barber Paradox: If the barber shaves every villager who does not shave himself, does the barber shave himself? Consider:

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

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. □

## 3. Probabilistic logic programs and their semantics

In this paper we focus on a particularly simple combination of logic programming and probabilities [21, 27]. 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 simply an atom $A$ associated with a probability value (assumed to be a rational number). For instance, $\mathbb{P}(\mathsf{insomnia} = \mathsf{true}) = 0.3$ is a probabilistic fact. If a probabilistic fact contains logical variables, we interpret it as the set of all grounded probabilistic facts obtained by substituting variables with constants in the Herbrand base. A probabilistic fact is assumed not to unify with any other atom in the head of a rule. Note that as a fact is the head of a rule, a probabilistic fact does not unify with any other fact.

The main probabilistic assumption is that all grounded probabilistic facts are independent. That is, there is a product probability measure over the probabilistic facts, such that $\mathbb{P}(A_1 = v_1, \ldots, A_m = v_m) = \prod_{i=1}^{m} \mathbb{P}(A_i = v_i)$ for any set of atoms $A_1, \ldots, A_m$ that appear in probabilistic facts, where each $v_i$ is either true or false. Suppose that for every fixed truth assignment over all grounded probabilistic facts we have a unique interpretation for the resulting normal logic program (that is, once we fix the probabilistic facts, all other atoms have their truth values fixed as true or false by the semantics). Then we see that the probabilities over probabilistic facts induce a probability distribution over all atoms. Sato's *distribution semantics*, originally proposed for definite programs, is exactly this probability distribution over all atoms [27]. Note that if a program is definite, then for each truth assignment for all probabilistic facts we have a single minimal model (stable model) that is exactly the well-founded model; thus for definite programs the distribution semantics is always unique.

If a program has an acyclic grounded dependency graph, it is said to be *acyclic* [1]. The well-founded semantics of an acyclic normal logic program has a unique interpretation for all atoms. Now if a PLP $\langle \mathbf{P}, \mathbf{PF}\rangle$ is such that $\mathbf{P}$ is acyclic, then for each truth assignment for grounded probabilistic facts the resulting normal logic program is acyclic as well. And then there is a unique distribution semantics for the program. The same is true for *stratified programs*; that is, programs where cycles in the grounded dependency

graph contain no negative edge (this is often referred to as *locally stratified* in the literature) [2]. The well-founded semantics of a stratified program is a unique interpretation for all atoms, hence $\langle \mathbf{P}, \mathbf{PF} \rangle$ has a unique distribution semantics if $\mathbf{P}$ is stratified. Note that an acyclic program is stratified.

To simplify the text, we say that a PLP is acyclic, stratified, etc, when its underlying normal logic program satisfies the property of interest.

In short: if a PLP is stratified, then it defines a unique distribution over all atoms; we refer to this distribution as the *distribution semantics* of the PLP.

Now if a logic program is non-stratified, then its well-founded semantics may be a partial interpretation, and some atoms may be left as undefined. One possible reaction is to focus attention only on programs that do not admit a partial interpretation given any truth assignment for all probabilistic facts — Riguzzi refers to these programs as *sound* ones [24]. Another path, taken by Lukasiewicz [16, 17] and later by Hadjichristodoulou and Warren [15] is to accommodate undefined values within the semantics.[1] However, the approach by Lukasiewicz is to leave the probability of any formula as undefined whenever the formula is undefined in any partial interpretation (to determine whether a formula is undefined or not in a particular partial interpretation, three-valued logic is used). Hence, when a formula gets a numeric probability, its truth value agrees across stable models; thus any probability calculations that are produced with this sort of well-founded semantics agree with a semantics based on stable models that is actually proposed by Lukasiewicz [17, Theorem 4.5]. That is, Lukasiewicz' proposal is more akin to the stable model semantics than to the well-founded semantics.

The approach by Hadjichristodoulou and Warren [15] is to allow probabilities over partial interpretations, thus allowing probabilities over atoms that are undefined. This is a bold proposal as far as interpretation is concerned, as we explain in Section 5. 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 language and semantics is named WF-PRISM by Hadjichristodoulou and Warren).

The goal of the remainder of this paper is to study the computational complexity of inferences under this well-founded semantics, and to discuss some of its conceptual difficulties. Before we plunge into this endeavor, we present some examples.

**Example 4.** First, take the rules in Example 1, and probabilistic fact $\mathbb{P}(\mathsf{r} = \mathsf{true}) = 0.3$. Under the well-founded semantics, $\mathbb{P}(\mathsf{p} = \mathsf{undefined}) = \mathbb{P}(\mathsf{q} = \mathsf{undefined}) = 0.7$. □

**Example 5.** Now consider the following extended version of Example 1, adapted from Example IV.1 by Hadjichristodoulou and Warren [15]. Consider:

$$\mathsf{cold} :- \mathsf{headache}, \mathsf{a}. \qquad \mathsf{cold} :- \mathbf{not}\ \mathsf{headache}, \mathbf{not}\ \mathsf{a}. \qquad \mathbb{P}(\mathsf{a} = \mathsf{true}) = 0.34.$$
$$\mathsf{headache} :- \mathsf{cold}, \mathsf{b}. \qquad \mathsf{headache} :- \mathbf{not}\ \mathsf{b}. \qquad \mathbb{P}(\mathsf{b} = \mathsf{true}) = 0.25.$$

There are four truth assignments for probabilistic facts, each with a product probability and induced (three-valued) truth assignments for cold and headache:

[1] Sato et al. also allow their distribution to be defined over undefined values, but instead of the well-founded semantics they prescribe Fitting's three-valued semantics. This is a weaker semantics than the well-founded one, and the literature on logic programming has consistently preferred the latter, as we do in this paper.

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

Consequently, by collecting probabilities, we have $\mathbb{P}(\mathsf{cold} = \mathsf{true}) = 0.255$, $\mathbb{P}(\mathsf{cold} = \mathsf{undefined}) = 0.165$, and $\mathbb{P}(\mathsf{cold} = \mathsf{false}) = 0.580$. And $\mathbb{P}(\mathsf{headache} = \mathsf{true}) = 0.750$, $\mathbb{P}(\mathsf{headache} = \mathsf{undefined}) = 0.165$, and $\mathbb{P}(\mathsf{headache} = \mathsf{false}) = 0.085$. □

**Example 6.** Consider Example 2, and suppose that $\mathsf{move}(a, b)$ is not a fact, but instead a probabilistic fact with associated assessment $\mathbb{P}(\mathsf{move}(a, b) = \mathsf{true}) = 0.5$. Then $\mathbb{P}(\mathsf{wins}(c) = \mathsf{true}) = \mathbb{P}(\mathsf{wins}(d) = \mathsf{false}) = 1$. We also have that $\mathbb{P}(\mathsf{wins}(a) = \mathsf{true}) = 0$, $\mathbb{P}(\mathsf{wins}(a) = \mathsf{undefined}) = \mathbb{P}(\mathsf{wins}(a) = \mathsf{false}) = 0.5$, and that $\mathbb{P}(\mathsf{wins}(b) = \mathsf{true}) = \mathbb{P}(\mathsf{wins}(b) = \mathsf{undefined}) = 0.5$, $\mathbb{P}(\mathsf{wins}(b) = \mathsf{false}) = 0$. □

**Example 7.** Consider Example 3, on the Barber Paradox. Suppose we add the probabilistic fact $\mathbb{P}(\mathsf{barber}(c) = \mathsf{true}) = 0.5$ and the fact $\mathsf{villager}(c)$.. With probability $0.5$, $\mathsf{shaves}(c, c)$ is false, and with probability $0.5$, $\mathsf{shaves}(c, c)$ is undefined. And $\mathbb{P}(\mathsf{shaves}(b, b) = \mathsf{undefined}) = 1$, while $\mathbb{P}(\mathsf{shaves}(a, a) = \mathsf{false}) = 1$. □

Given that well-founded semantics deals with interpretations containing classical negation $\neg$, it would perhaps be more elegant to have classical negation in the language from the outset. In fact, that would clarify the meaning of probabilities attached to probabilistic facts: $\mathbb{P}(A = \mathsf{true}) = \alpha$ would mean that $A$ is true with probability $\alpha$ and $\neg A$ is true with probability $1 - \alpha$, leaving no probability mass to the undefined value. It turns out that, once we allow cycles in a normal logic program, we can "simulate" classical negation of atoms [13]. In short, we do so by replacing each $\neg A$ by a fresh atom $A'$, and by imposing the constraint that $A \wedge A'$ cannot be both true; this constraint can be built with a cyclic rule. Thus our results on complexity do not change if we allow classical negation to be placed directly in front of atoms.

## 4. The complexity of inference under the well-founded semantics

We wish to investigate the complexity of *probabilistic* inference under the well-founded semantics. To do so, we need a number of definitions that we now quickly review.

We employ usual complexity classes P, NP, EXP, NEXP, and related ones, and we use oracle Turing machines [20]. The *polynomial hierarchy* is the class of languages $\bigcup_i \Delta_i^p = \bigcup_i \Pi_i^p = \bigcup_i \Sigma_i^p$, where $\Delta_i^p = \mathsf{P}^{\Sigma_{i-1}^p}$, $\Pi_i^p = \mathsf{co}\Sigma_i^p$, $\Sigma_i^p = \mathsf{NP}^{\Sigma_{i-1}^p}$ and $\Sigma_0^p = \mathsf{P}$. The complexity class #P is the class of integer-valued functions computed by a counting Turing machine in polynomial time; a counting Turing machine is a standard nondeterministic Turing machine that prints in binary notation, on a separate tape, the number of accepting computations induced by the input [30]. The class $\#\Sigma_k^p$ contains the integer-valued functions computed by a counting Turing machine with access to a $\Sigma_k^p$ oracle (note: $\#\Sigma_0^p = \#\mathsf{P}$). We are interested in the computation of probabilities, so we are not really dealing with integer-valued outputs. Hence we resort to *weighted reductions* [5]; that is, parsimonious reductions (Karp reductions that preserve the number of accepting paths) scaled by a polynomial-time computable positive rational number. Using such reductions one can normalize the output of counting Turing machines, thus obtaining probabilities.

For any complexity class #C in the polynomial counting hierarchy, we say that a problem is #C-hard if any problem in #C can be reduced to it by a weighted reduction. If a problem is #C-hard and can be solved with a polynomial number of pre-processing steps followed by one call to a #C oracle and a multiplication by a rational obtained with polynomial effort, then the problem is said to be #C-equivalent (an alternative definition is advocated by Ref. [11]). For instance, the complexity of inferences in Bayesian networks is #P-complete [25]. We also use the class #EXP, which contains functions computed by counting Turing machines in exponential time [19] (this is not equal to the homonymous class defined by Valiant [30]). Hardness and equivalence for #EXP are defined as for #P, with polynomial replaced by exponential.

Our inference problem is formalized as follows:

**Input:** a PLP whose probabilities are rational numbers, and a truth assignment $\mathbf{Q}$ to some ground atoms in the (finite) Herbrand base.
**Output:** the value of (the rational number) $\mathbb{P}(\mathbf{Q})$.

We refer to this complexity as the *inference complexity* of PLPs. In practice it may happen that the program is small compared to the query (similarly to what happens in database theory, where the database is typically much larger than then instructions specifying records of interest). We thus consider the following problem, which we refer as the *query complexity* of PLPs [8]:

**Input:** a truth assignment $\mathbf{Q}$ to some ground atoms in the (finite) Herbrand base (the PLP is fixed, and not considered part of the input).
**Output:** the value of (the rational number) $\mathbb{P}(\mathbf{Q})$.

We can now state our main result. We examine propositional and relational programs, and within the latter we look at programs with a bound on predicate arity. Note that a bound on predicate arity forces each predicate to have a polynomial number of groundings, but the grounding of the program may still be exponential (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).

**Theorem 1.** *The inference complexity of* PLP*s is* #EXP*-complete; it is* #NP*-complete if the* PLP *has a bound on the arity of its predicates; it is* #P*-complete if the* PLP *is propositional. The query complexity of* PLP*s is* #P*-complete.*

*Proof.* Consider first propositional PLPs. Such a PLP can encode any Bayesian network over binary variables [21], so inference is #P-hard. Membership follows from the fact that logical inference with propositional normal logic programs under the well-founded semantics is in P [9]; hence we can have a counting Turing machine that guesses a truth assignment for probabilistic facts and then produces in polynomial time the truth assignment for all atoms (this counting Turing machine produces an integer that yields the desired probability after division by a constant).
Consider now PLPs with logical variables. Such a PLP can encode any "enhanced" plate model as defined in Ref. [6], so inference complexity is #EXP-hard. The enhanced plates that must be encoded are acyclic probabilistic models whose nodes are relations, and whose probabilities are specified either through probabilistic facts or as propositional Boolean operators or as existential quantifiers; such an acyclic model can be reproduced

by an acyclic program (with predicates containing logical variables) and the Clark completion of the program [31] then produces the needed existential quantifiers. Membership follows from the fact that inference in normal logic programs under the well-founded semantics is in EXP [9] (using the same reasoning as for propositional programs). The hardness of query complexity also follows from the parallel result for plate models [6], and membership follows from the fact that data complexity of normal logic programs under the well-founded semantics is in P [9].
Now consider PLPs with a bound on the arity of predicates. Hardness for #NP follows from the fact that inference complexity of PLPs under the stable model semantics is #NP-hard even for stratified programs [7], noting that for stratified programs the stable model and the well-founded semantics agree. Membership follows from the fact that logical inference is in $\mathsf{P}^{\mathsf{NP}}$, as proved in Theorem 2; thus inference can be produced by a counting Turing machine that nondeterministically selects the truth assignments for all probabilistic facts and then runs logical inference by runing polynomial operations and calls to an NP oracle (plus a final division by a constant). □

The proof of Theorem 1, 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 2.** *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* $\mathsf{P}^{\mathsf{NP}}$*-complete.*

*Proof.* Hardness follows from the hardness of logical inference with stratified programs under the stable model semantics [12]. 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 $\mathsf{P}^{\mathsf{NP}}$ as desired. □

To conclude this section, we note that, for the stable model semantics, the inference complexity of propositional PLPs is already in #NP, while the inference complexity of PLPs with a bound on predicate arity is in $\#\mathsf{NP}^{\mathsf{NP}}$, and the query complexity is in

#NP [7]. Thus the well-founded semantics does lead to smaller complexity classes; this is consistent with the fact that logical inference with the well-founded semantics leads to smaller complexity classes than logical inference with the stable model semantics [9].

## 5. The semantics of the well-founded semantics

The well-founded semantics of PLPs is attractive because it offers a unique semantics for every probabilistic logic program. To some extent, the semantics is conceptually simple — at least for someone who has mastered the well-founded semantics for normal logic programs. However, the meaning of the well-founded semantics deserves some attention.

One problem with the well-founded semantics is that it is somewhat weak, and repeatedly noted in the literature. Consider the program [31]:

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

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. Note that these two interpretations are exactly the stable models: one contains a and $\neg$b, the other contains $\neg$a and b.

However weak, this sort of semantics may be appropriate sometimes: consider for instance the Barber Paradox, where the well-founded semantics does not determine whether the barber shaves himself, but still produces sensible inference for other villagers. In other examples the well-founded semantics may be open to criticism in its reliance on three-valued logic, as it invites controversy on the meaning of undefined. It is 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 [33, Section 1.2.1.2]. In fact the vagaries of three-valued logic have received attention not only in philosophical inquiry [4, 18], but in the practical development of databases [10, 26].

In any case, we do not want to repeat the old and unresolved debate on three-valued logic here. Our point is that, in the presence of probabilities, matters become even murkier. The problem 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 Example 5. What does it mean to say that $\mathbb{P}(\mathsf{headache} = \mathsf{undefined}) = 0.165$? Supposedly probability is here to tell us the odds of true and false; by learning the probability of undefined, the next question should be about the probability of headache to be true when one is saying that it is undefined. In fact, one might 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. To emphasize the difficulty in interpretation, suppose we add to Example 5 the simple rule

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

and one asks for $\mathbb{P}(\mathsf{c} = \mathsf{false}|\mathsf{cold} = \mathsf{undefined})$. Should this number really be 1, as obtained through the well-founded semantics, or should it simply be the interpreted as a question about $\mathbb{P}(\mathsf{c} = \mathsf{false})$, given that nothing of substance is said about cold? By looking at such examples, one can understand Riguzzi's refusal in dealing with undefined

values: as he writes, "the uncertainty should be handled by the choices [that is, by the probabilistic facts] rather than by the semantics of negation."

As a final example, consider again the Barber Paradox. As noted before, it makes some sense, for the *logical program* in Example 3, to obtain undefined for $\mathsf{shaves}(b, b)$. But for the probabilistic program in Example 7, the meaning of $\mathbb{P}(\mathsf{shaves}(c, c) = \mathsf{undefined}) = 0.5$ is quite hard to ascertain. We should expect, given that there is uncertainty, a probability that $\mathsf{shaves}(c, c)$ is true, and a probability that it is false; instead we are served with the probability that it is undefined.

A difficulty here is that undefined values may appear due to a variety of situations: programs may be inconsistent (as it happens in Example 5), may fail to have a clear meaning (as in the Barber Paradox), or simply may have various possible meanings (for instance, various stable models). In the first two cases, it is even surprising that one would assign probabilities to the inconsistent or contradictory cases. In the latter case, probabilities may be contemplated, but then one may be surprised to see a mixture of probabilities, presumably used to encode some subjective uncertainty, and undefined values that also encode subjective uncertainty (we do not know the right model). The interpretation of the various possible meanings of undefined, already difficult in three-valued logic, is magnified by the challenges in interpreting probabilities.

## 6. Conclusion

In this paper we have studied the well-founded semantics of probabilistic normal logic programs as advanced by Hadjichristodoulou and Warren [15]. Their proposal seems to be the one that most strongly stays close to the (logical) well-founded semantics. Hence it inherits an important advantage from the (logical) well-founded semantics: every probabilistic logic program has a well-founded semantics.

We have examined the complexity of inference with probabilistic logic programs under the well-founded semantics, and have clarified its properties in a number of directions, ranging from propositional to relational programs, and from inference to query complexity. These results should be useful in guiding the comparison between the well-founded semantics and rival approaches.

We have also discussed the interpretation of the well-founded semantics for probabilistic logic programs. We found that such an interpretation is not an easy matter, and in many programs the semantics seems to unduly put together various notions of uncertainty. We suggest that more study is needed to isolate those programs where undefined values are justified and can be properly mixed with probabilities. For instance, we feel that such a mix may be useful in dealing with programs that have many stable models, where the well-founded semantics may be taken as an approximation of the set of possible semantics. Such a discussion must be the subject of future research.

## 7. Acknowledgement

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

## References


[1] Krzysztof R. Apt and Marc Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

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

[4] Merrie Bergmann. *An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems*. Cambridge University Press, 2008.

[5] Andrei Bulatov, Martin Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby. The complexity of weighted and unweighted #CSP. *Journal of Computer and System Sciences*, 78:681–688, 2012.

[6] Fabio Gagliardi Cozman and Denis Deratani Mauá. The complexity of plate probabilistic models. In *Scalable Uncertainty Management*, volume 9310 of *LNCS*, pages 36–49. Springer, 2015.

[7] Fabio Gagliardi Cozman and Denis Deratani Mauá. The structure and complexity of credal semantics. *Workshop on Probabilistic Logic Programming*, pages 3-14, 2016.

[8] Fabio Gagliardi Cozman and Denis Deratani Mauá. Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity. In *AAAI Conference on Artificial Intelligence*, 2015.

[9] Evgeny Dantsin, Thomas Eiter, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[10] Christopher J. Date. Database in depth: Relational theory for practitioners, 2005.

[11] Cassio Polpo de Campos, Georgios Stamoulis, and Dennis Weyland. A structured view on weighted counting with relations to quantum computation and applications. Technical Report 133, Electronic Colloquium on Computational Complexity, 2013.

[12] Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, 5:123–165, 2007.

[13] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwalner. Answer set programming: a primer. In *Reasoning Web*, pages 40–110. Springer-Verlag, 2009.

[14] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of Int. Logic Programming Conference and Symposium*, volume 88, pages 1070–1080, 1988.

[15] Spyros Hadjichristodoulou and David S. Warren. Probabilistic logic programming with well-founded negation. In *Int. Symposium on Multiple-Valued Logic*, pages 232–237, 2012.

[16] Thomas Lukasiewicz. Probabilistic description logic programs. In *ECSQARU*, pages 737–749, Spain, July 2005. Springer.

[17] Thomas Lukasiewicz. Probabilistic description logic programs. *Int. Journal of Approximate Reasoning*, 45(2):288–307, 2007.

[18] Grzegorz Malinowski. Many-valued logic and its philosophy. In Dov M. Gabbay and John Woods, editors, *Handbook of the History of Logic - Volume 8*, pages 13–94. Elsevier, 2007.

[19] Christos H. Papadimitriou. A note on succinct representations of graphs. *Information and Control*, 71:181–185, 1986.

[20] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing, 1994.

[21] David Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[22] David Poole. The Independent Choice Logic and beyond. In L. De Raedt, Paolo Frasconi, Kristian Kersting, Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming*, *Lecture Notes in Computer Science 4911*, pages 222–243. Springer, 2008.

[23] Teodor Przymusinski. Every logic program has a natural stratification and an iterated least fixpoint model. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 11–21, 1989.

[24] Fabrizio Riguzzi. The distribution semantics is well-defined for all normal programs. In Fabrizio Riguzzi and Joost Vennekens, editors, *Int. Workshop on Probabilistic Logic Programming*, volume 1413 of *CEUR Workshop Proceedings*, pages 69–84, 2015.

[25] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

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

[27] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Int. Conference on Logic Programming*, pages 715–729, 1995.

[28] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[29] Taisuke Sato, Yoshitaka Kameya, and Neng-Fa Zhou. Generative modeling with failure in PRISM. In *Int. Joint Conference on Artificial Intelligence*, pages 847–852, 2005.

[30] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):410–421, 1979.

[31] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery*, 38(3):620–650, 1991.

[32] Allen Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47:185–221, 1993.

[33] Mark Wallace. Tight, consistent, and computable completions for unrestricted logic programs. *Journal of Logic Programming*, 15:243–273, 1993.