IFIG
RESEARCH
REPORT

INSTITUT FÜR INFORMATIK

PROBABILISTIC AND TRUTH-FUNCTIONAL MANY-VALUED LOGIC PROGRAMMING

Thomas Lukasiewicz

IFIG RESEARCH REPORT 9809
DECEMBER 1998

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

IFIG Research Report
IFIG Research Report 9809, December 1998

Probabilistic and Truth-Functional Many-Valued Logic Programming

Thomas Lukasiewicz*

Abstract. We introduce probabilistic many-valued logic programs in which the implication connective is interpreted as material implication. We show that probabilistic many-valued logic programming is computationally more complex than classical logic programming. More precisely, some deduction problems that are P-complete for classical logic programs are shown to be co-NP-complete for probabilistic many-valued logic programs. We then focus on many-valued logic programming in $\mathrm{Pr}_n^\star$ as an approximation of probabilistic many-valued logic programming. Surprisingly, many-valued logic programs in $\mathrm{Pr}_n^\star$ have both a probabilistic semantics in probabilities over a set of possible worlds and a truth-functional semantics in the finite-valued Łukasiewicz logics $\mathrm{Ł}_n$. Moreover, many-valued logic programming in $\mathrm{Pr}_n^\star$ has a model and fixpoint characterization, a proof theory, and computational properties that are very similar to those of classical logic programming. We especially introduce the proof theory of many-valued logic programming in $\mathrm{Pr}_n^\star$ and show its soundness and completeness.

*Institut für Informatik, Universität Gießen, Arndtstraße 2, D-35392 Gießen, Germany
Tel: +49-641-99-32153, Fax: +49-641-99-32149, E-mail: lukasiewicz@informatik.uni-giessen.de

1 Introduction

Recently, I was posed the following question: "I would like to add uncertainty to my knowledge-base system. Which formalism do you suggest me?". I was quick to reply that from the semantic point of view, I would suggest a probabilistic framework, but that such an approach would surely slow down the system significantly. Since this reply did not seem to be satisfactory, I was posed another question: "What about the finite-valued Łukasiewicz logics?"

This paper justifies my reply to the first question and also gives a very surprising answer to the second one.

More precisely, we start by presenting probabilistic many-valued logic programs in which the implication connective is interpreted as material implication. We then show that probabilistic many-valued logic programming in this framework is computationally more complex than classical logic programming. More precisely, some deduction problems that are P-complete for classical logic programs are shown to be co-NP-complete for probabilistic many-valued logic programs (see also [28], [29], and [30] for other work on the subtleties and the computational complexity of probabilistic deduction).

We then focus on many-valued logic programming in $\mathrm{Pr}_n^\star$ as an approximation of probabilistic many-valued logic programming. Crucially, many-valued logic programming in $\mathrm{Pr}_n^\star$ has a model and fixpoint characterization and a proof theory that are very similar to those of classical logic programming. Furthermore, special cases of many-valued logic programming in $\mathrm{Pr}_n^\star$ have the same computational complexity like their classical counterparts.

Surprisingly (and at first sight even paradoxically), many-valued logic programs in $\mathrm{Pr}_n^\star$ have both a probabilistic semantics in probabilities over a set of possible worlds and a truth-functional semantics in the finite-valued Łukasiewicz logics $\mathrm{Ł}_n$. That is, many-valued logic programming in $\mathrm{Pr}_n^\star$ lies in the intersection between probabilistic logics and truth-functional many-valued logics.

The literature already contains quite extensive work on probabilistic and on truth-functional many-valued logic programming separately. However, to the best of our knowledge, an integration of both has never been studied so far.

Probabilistic propositional logics and their various dialects are thoroughly studied in the literature (see, for example, [36] and [13]). Their extensions to probabilistic first-order logics can be classified into first-order logics in which probabilities are defined over a set of possible worlds and those in which probabilities are given over the domain (see, for example, [3], [4], and [20]). The first ones are suitable for representing degrees of belief, while the latter are appropriate for describing statistical knowledge. The same classification holds for probabilistic logic programming (see, for example, [33], [34], [35], [30], and [32]).

Many approaches to truth-functional finite-valued logic programming are restricted to three or four truth values (see, for example, [23], [14], [15], [6], and [11]). Among these approaches, the one closest in spirit to many-valued logic programming in $\mathrm{Pr}_n^\star$ is perhaps the three-valued approach in [23].

Many-valued logic programming in $\mathrm{Pr}_n^\star$ is closely related to van Emden's infinite-valued quantitative deduction [39]. In detail, many-valued logic programming in $\mathrm{Pr}_n^\star$ is an approximation of probabilistic logic programming under the material implication, while van Emden's quantitative deduction can be understood as an approximation of probabilistic logic programming under the conditional probability implication [30].

Furthermore, many-valued logic programming in $\mathrm{Pr}_n^\star$ is related to the important work on generalized annotated logic programming [21] and to signed formula logic programming [26].

Many-valued logic programming in $\Pr_n^\star$ itself was initiated in [27], where we introduced many-valued first-order logics with probabilistic semantics and already presented a model and fixpoint characterization.

In this paper, many-valued first-order logics with probabilistic semantics are now analyzed more deeply from the logic programming viewpoint. The main new contributions of this paper can be summarized as follows:

- We introduce a probabilistic approach to many-valued logic programs in which the implication connective is interpreted as material implication.
- We show that probabilistic many-valued logic programming is computationally more complex than classical logic programming. That is, some deduction problems that are P-complete for classical logic programs are shown to be co-NP-complete for probabilistic many-valued logic programs.
- We show that probabilistic many-valued logic programming is approximated by many-valued logic programming in $\Pr_n^\star$ as introduced in [27].
- We present a proof theory for many-valued logic programming in $\Pr_n^\star$ and show its soundness and completeness.

The rest of this paper is organized as follows. In Section 2, we focus on probabilistic many-valued logic programming. Section 3 concentrates on its approximation by many-valued logic programming in $\Pr_n^\star$. In Section 4, we summarize the main results and give an outlook on future research.

2 Probabilistic Many-Valued Logic Programming

In this section, we introduce probabilistic many-valued logic programs. After giving an illustrative example, we then analyze some properties and the computational complexity of probabilistic many-valued logic programming.

2.1 Technical Preliminaries

We now briefly summarize how classical first-order logics can be given a probabilistic n-valued semantics with $n \geq 3$ in which probabilities are defined over a set of possible worlds. We basically follow the important work of Halpern [20], which we restrict and adapt to our needs in the n-valued setting.

Let $TV = \{0, \frac{1}{n-1}, \frac{2}{n-1}, \ldots, 1\}$ with $n \geq 3$ denote the set of *truth values*. Note that the classical truth values **false** and **true** correspond to 0 and 1, respectively. Let Φ be a first-order vocabulary that contains a set of function symbols and a set of predicate symbols (as usual, *constant symbols* are function symbols of arity zero; we say Φ is *function-free* if it does not contain any function symbols of arity greater than zero). Let $\mathcal{X}$ be a set of *object* and *bound variables*. Note that object variables represent elements from a certain domain, while bound variables describe truth values from TV.

We define *object terms* by induction as follows. An object term is an object variable from $\mathcal{X}$ or an expression of the kind $f(t_1, \ldots, t_k)$, where f is a function symbol of arity $k \geq 0$ from Φ and $t_1, \ldots, t_k$ are object terms. A *bound term* is a truth value from TV or a bound variable from $\mathcal{X}$. We define *formulas* by induction as follows. If p is a predicate symbol of arity $k \geq 0$ from Φ and

$t_1, \ldots, t_k$ are object terms, then $p(t_1, \ldots, t_k)$ is a formula (called *atomic formula*). If F and G are formulas, then $\neg F$, $(F \wedge G)$, $(F \vee G)$, and $(F \leftarrow G)$ are formulas. If F is a formula and x is an object variable from $\mathcal{X}$, then $\forall x\, F$ and $\exists x\, F$ are formulas. An *n-valued formula* is an expression $\mathsf{tv}(F) \geq t$, where F is a classical formula and t is a bound term.

An *interpretation* $I = (D, \pi)$ consists of a nonempty set D, called *domain*, and a mapping π that assigns to each function symbol from Φ a function of right arity over D and to each predicate symbol from Φ a predicate of right arity over D. A *variable assignment* σ is a mapping that assigns to each object variable from $\mathcal{X}$ an element from D and to each bound variable from $\mathcal{X}$ a truth value from TV. For an object variable x from $\mathcal{X}$ and an element d from D, we write $\sigma[x/d]$ to denote the variable assignment that is identical to σ except that it assigns d to x (for a bound variable x from $\mathcal{X}$ and a truth value c from TV, the notation $\sigma[x/c]$ has an analogous meaning). The variable assignment σ is by induction extended to all object and bound terms by defining $\sigma(f(t_1, \ldots, t_k)) = \pi(f)(\sigma(t_1), \ldots, \sigma(t_k))$ for all object terms $f(t_1, \ldots, t_k)$ and $\sigma(c) = c$ for all truth values c from TV. The *truth* of formulas F in I under σ, denoted $I \models_\sigma F$, is inductively defined as follows:

- $I \models_\sigma p(t_1, \ldots, t_k)$ iff $(\sigma(t_1), \ldots, \sigma(t_k)) \in \pi(p)$.
- $I \models_\sigma \neg F$ iff not $I \models_\sigma F$, and $I \models_\sigma (F \wedge G)$ iff $I \models_\sigma F$ and $I \models_\sigma G$.
- $I \models_\sigma \forall x\, F$ iff $I \models_{\sigma[x/d]} F$ for all $d \in D$.
- The truth of the remaining formulas in I under σ is defined by expressing $\vee$, $\leftarrow$, and $\exists$ in terms of $\neg$, $\wedge$, and $\forall$ as usual.

A formula F *is true* in I, or I is a *model* of F, denoted $I \models F$, iff F is true in I under all variable assignments σ.

A *probabilistic interpretation* (also Pr_n*-interpretation*) Pr is a triple $(D, \mathcal{I}, \mu)$, where D is a nonempty set (called *domain*), $\mathcal{I}$ is a set of classical interpretations over D (which are called *possible worlds*) such that $\pi_i(f) = \pi_j(f)$ for all function symbols f from Φ and all interpretations $(D, \pi_i), (D, \pi_j) \in \mathcal{I}$, and μ is a mapping from $\mathcal{I}$ to the set of truth values TV such that all $\mu(I)$ with $I \in \mathcal{I}$ sum up to 1. The *truth value* $Pr_\sigma(F)$ of a formula F in the Pr_n-interpretation Pr under a variable assignment σ is defined as follows (if F does not contain any variables, then $Pr_\sigma(F)$ is abbreviated by $Pr(F)$):

$$Pr_\sigma(F) \;=\; \sum_{I \in \mathcal{I},\; I \models_\sigma F} \mu(I)\,. \tag{1}$$

An n-valued formula $\mathsf{tv}(F) \geq t$ *is true* in Pr under σ iff $Pr_\sigma(F) \geq \sigma(t)$. An n-valued formula P *is true* in Pr, or Pr is a *model* of P, denoted $Pr \models P$, iff P is true in Pr under all variable assignments σ. The Pr_n-interpretation Pr is a *model* of a set of n-valued formulas $\mathcal{P}$, denoted $Pr \models \mathcal{P}$, iff Pr is a model of all n-valued formulas in $\mathcal{P}$. The set of n-valued formulas $\mathcal{P}$ is *satisfiable* iff a model of $\mathcal{P}$ exists. The n-valued formula P is a *logical consequence* of $\mathcal{P}$, denoted $\mathcal{P} \models P$, iff each model of $\mathcal{P}$ is also a model of P.

For an n-valued formula $\mathsf{tv}(F) \geq c$ with a truth value c from TV and a set of n-valued formulas $\mathcal{P}$, let $\boldsymbol{c}$ denote the set of all truth values $Pr_\sigma(F)$ in models Pr of $\mathcal{P}$ under variable assignments σ. We verify easily that $\mathsf{tv}(F) \geq c$ is a logical consequence of $\mathcal{P}$ iff $c \leq \min \boldsymbol{c}$. Hence, we get a natural notion of tightness for logical consequences: the n-valued formula $\mathsf{tv}(F) \geq c$ is a *tight logical consequence* of $\mathcal{P}$, denoted $\mathcal{P} \models_{tight} \mathsf{tv}(F) \geq c$, iff $c = \min \boldsymbol{c}$.

A *Herbrand* $\Pr_n$*-interpretation* $(\mathcal{I}, \mu)$ consists of a set $\mathcal{I}$ of classical Herbrand interpretations over Φ (that is, subsets of the Herbrand base HB_Φ over Φ) and a mapping μ from $\mathcal{I}$ to TV such that all $\mu(I)$ with $I \in \mathcal{I}$ sum up to 1.

Terms, formulas, n-valued formulas, and sets of n-valued formulas are *ground* iff they do not contain any variables. The notions of substitutions, ground substitutions, instances of formulas, and ground instances of formulas are defined as usual. The last two are assumed to be canonically extended to n-valued formulas. Finally, we also adopt the usual conventions to eliminate parentheses.

2.2 Many-Valued Logic Programs

We now introduce probabilistic many-valued logic programs. We start by defining many-valued program clauses, which are special many-valued formulas (thus, they inherit their syntax and semantics from many-valued formulas).

An *n-valued program clause* is an n-valued formula $\mathsf{tv}(H \vee \neg B_1 \vee \cdots \vee \neg B_k) \geq c$, where $H, B_1, \ldots, B_k$ with $k \geq 0$ are atomic formulas and c is a truth value from TV. It is abbreviated by $(H \leftarrow B_1, \ldots, B_k)[c, 1]$. Note that all object variables in an n-valued program clause are implicitly universally quantified. An *n-valued logic program* is a finite set of n-valued program clauses.

Hence, many-valued program clauses are in the following sense an extension of classical program clauses. Many-valued program clauses may be assigned any subset $\{c, \ldots, 1\}$ of TV as a range for their probabilistic truth value, while classical program clauses always have the classical truth value **true**.

Many-valued program clauses can be classified into facts and rules as follows. *Facts* are many-valued program clauses of the kind $(H \leftarrow\ )[c, 1]$, while *rules* are many-valued program clauses of the form $(H \leftarrow B_1, \ldots, B_k)[c, 1]$ with $k > 0$. They can also be divided into logical and purely many-valued program clauses: *logical program clauses* are of the kind $(H \leftarrow B_1, \ldots, B_k)[1, 1]$, while *purely many-valued program clauses* have the form $(H \leftarrow B_1, \ldots, B_k)[c, 1]$ with $c < 1$.

Next, we introduce many-valued queries, answer substitutions, and answers. These definitions are based on the notions of logical consequence and of tight logical consequence as introduced in the previous subsection.

An *n-valued query* to an n-valued logic program $\mathcal{P}$ is an expression of the form $\exists(A_1, \ldots, A_l)[t, 1]$, where $A_1, \ldots, A_l$ with $l \geq 1$ are atomic formulas and t is a bound term. An n-valued query is *object-ground* iff it does not contain any object variables. Given an n-valued query $Q_c = \exists(A_1, \ldots, A_l)[c, 1]$ with $c \in TV$, we are interested in its *correct answer substitutions*, which are substitutions θ such that $\mathcal{P} \models \mathsf{tv}((A_1 \wedge \cdots \wedge A_l)\theta) \geq c$ and that θ acts only on variables in Q_c. The *correct answer* for Q_c is Yes if a correct answer substitution exists and No otherwise. Given an n-valued query $Q_x = \exists(A_1, \ldots, A_l)[x, 1]$ with $x \in \mathcal{X}$, we are interested in its *tight answer substitutions*, which are substitutions θ such that $\mathcal{P} \models_{\mathit{tight}} \mathsf{tv}((A_1 \wedge \cdots \wedge A_l)\theta) \geq x\theta$, that θ acts only on variables in Q_x, and that $x\theta$ is a truth value from TV. Note that such n-valued queries Q_x always have a tight answer substitution. In particular, object-ground n-valued queries Q_x always have exactly one tight answer substitution.

In the sequel, we use *probabilistic many-valued logic programming* as a synonym for the problem of deciding whether Yes is the correct answer for a given ground many-valued query to a many-valued logic program.

2.3 Example

Let us assume that John wants to pick up Mary after she stopped working. To do so, he must drive from his home to her office. However, he left quite late. So, he is wondering if he can still reach her in time. Unfortunately, since it is rush hour, it is very probable that he runs into a traffic jam. Now, John has the following knowledge at hand: given a road from R to S, the probability that he can reach S through R without running into a traffic jam is greater than 70% (a), given a road in the south of the town, he knows that this probability is even greater than 90% (b). A friend just called him and told him about some roads without any significant traffic (c). Furthermore, he clearly knows that if he can reach S through T and T through R, both without running into a traffic jam, then he can also reach S through R without running into a traffic jam (d). This knowledge can be expressed as follows (R, S, and T are object variables):

$$\begin{array}{ll}
\text{(a)} & (re(R,S) \leftarrow ro(R,S))[.7,1] \\
\text{(b)} & (re(R,S) \leftarrow ro(R,S), so(R,S))[.9,1] \\
\text{(c)} & (re(R,S) \leftarrow ro(R,S), ad(R,S))[1,1] \\
\text{(d)} & (re(R,S) \leftarrow re(R,T), re(T,S))[1,1]\,.
\end{array}$$

John is wondering whether he can reach Mary's office from his home, such that the probability of him running into a traffic jam is smaller than 1%. This can be expressed by the ground many-valued query $\exists(re(h,o))[.99,1]$. His wondering about which places are reachable from his home, such that the probability of him running into a traffic jam is smaller than 20%, can be described by the many-valued query $\exists(re(h,U))[.8,1]$, where U is an object variable. Finally, his wondering about the greatest lower bound for the probability of reaching the office, without running into a traffic jam, can be expressed by the many-valued query $\exists(re(h,o))[X,1]$, where X is a bound variable.

Of course, the correct answers for the first two queries and the tight answer substitution for the third query depend on the concrete knowledge of the roads, of the roads in the south of the town, and of the roads that John's friend was talking about. Some self-explaining many-valued program clauses that describe this knowledge could be given as follows (h, a, b, and o are 0-ary function symbols; the fifth clause describes the fact that John is not sure anymore whether or not his friend was talking about the road from a to b):

$$\begin{array}{l}
(ro(h,a) \leftarrow\ )[1,1], (ro(a,b) \leftarrow\ )[1,1], (ro(b,o) \leftarrow\ )[1,1] \\
(ad(h,a) \leftarrow\ )[1,1], (ad(a,b) \leftarrow\ )[.8,1], (so(b,o) \leftarrow\ )[1,1]\,.
\end{array}$$

Let $n = 101$ and let $\mathcal{P}$ denote the n-valued logic program that contains all the n-valued program clauses of this section. The correct answer for the n-valued query $\exists(re(h,o))[.99,1]$ to $\mathcal{P}$ is No, whereas the correct answer for $\exists(re(h,U))[.8,1]$ to $\mathcal{P}$ is Yes (note that all the correct answer substitutions for $\exists(re(h,U))[.8,1]$ to $\mathcal{P}$ are given by $\{U/a\}$ and $\{U/b\}$). Finally, the unique tight answer substitution for $\exists(re(h,o))[X,1]$ to $\mathcal{P}$ is given by $\{X/.7\}$.

2.4 Important Properties

Classical logic programs have the nice property that they are always satisfiable. We now show that many-valued logic programs have the same nice property.

Theorem 2.1 *Every n-valued logic program is satisfiable.*

Proof. The Herbrand Pr_n-interpretation $(\mathcal{I}, \mu)$ that is defined by $\mathcal{I} = \{HB_\Phi\}$ and $\mu(HB_\Phi) = 1$ is a model of every n-valued logic program. $\Box$

Another nice property of many-valued logic programs is that ground many-valued formulas are logically entailed in Pr_n-interpretations iff they are logically entailed in Herbrand Pr_n-interpretations, which follows from the next theorem (hence, if we want to compute correct answers for ground many-valued queries, then we can restrict our attention to Herbrand Pr_n-interpretations).

Theorem 2.2 *Let $\mathcal{P}$ be a set of quantifier-free n-valued formulas and let F be a ground formula. For each Pr_n-interpretation Pr with $Pr \models \mathcal{P}$, there exists a Herbrand Pr_n-interpretation Pr' with $Pr' \models \mathcal{P}$ and $Pr(F) = Pr'(F)$.*

Proof. Let $Pr = (D, \mathcal{I}, \mu)$. For each $I \in \mathcal{I}$ let $H(I)$ be the set of all $A \in HB_\Phi$ with $I \models A$. Let $Pr' = (\mathcal{I}', \mu')$, where $\mathcal{I}' = \{H(I) \mid I \in \mathcal{I}\}$ and $\mu'(I')$, for all $I' \in \mathcal{I}'$, is the sum of all $\mu(J)$ with $J \in \mathcal{I}$ and $H(J) = I'$. By induction on the length of formulas, it can now be shown that I and $H(I)$, for all $I \in \mathcal{I}$, are models of the same ground formulas. Hence, we also get $Pr(G) = Pr'(G)$ for all ground formulas G. Thus, Pr' is a model of $\mathcal{P}$ with $Pr(F) = Pr'(F)$. $\Box$

2.5 Computational Complexity

Probabilistic many-valued logic programming in its full generality is immediately undecidable, since it is a generalization of classical logic programming.

We now analyze the computational complexity of two special cases of decidable probabilistic many-valued logic programming. The first one is a generalization of propositional logic programming, while the second one generalizes the decision problem that defines the data complexity of datalog.

These two special cases of decidable probabilistic many-valued logic programming are of special interest, since their classical counterparts have the nice property that they are P-complete (see, for example, [8] for a survey).

Crucially, the P-completeness does not carry over to the two probabilistic many-valued generalizations, which are now shown to be co-NP-complete.

Theorem 2.3 *The problem of deciding whether* Yes *is the correct answer for a ground n-valued query $\exists(A_1, \ldots, A_l)[c, 1]$ to a ground n-valued logic program $\mathcal{P}$ is co-NP-complete.*

Proof. We subsequently show that the problem of deciding whether No is the correct answer for $\exists(A_1, \ldots, A_l)[c, 1]$ to $\mathcal{P}$ is NP-complete.

The problem is in NP, since we just have to guess an n-valued Herbrand interpretation $(\mathcal{I}, \mu)$ (note that $\mu(I) > 0$ for maximally $n-1$ classical Herbrand interpretations $I \in \mathcal{I}$ and that all $I \in \mathcal{I}$ just have to be subsets of the set of all ground atomic formulas in $\mathcal{P}$) and check whether $(\mathcal{I}, \mu)$ is a model of $\mathcal{P}$ and not a model of $\mathsf{tv}(A_1 \wedge \cdots \wedge A_l) \geq c$. This can be done by a nondeterministic algorithm in polynomial time (note that we assume a fixed number of truth values n).

To show NP-hardness for $n = 3$, we give a polynomial reduction from the NP-complete problem of hypergraph 2-colorability [17]. Let $\mathcal{C}$ be a set of nonempty subsets of a finite set S. Let the vocabulary Φ be the least set that contains the 0-ary predicate symbols d and s for each $s \in S$. We now construct a ground 3-valued logic program $\mathcal{P}$ as follows. We start by initializing $\mathcal{P}$ with $\emptyset$. For each $s \in S$, we increase $\mathcal{P}$ by $(s \leftarrow\ )[\frac{1}{2}, 1]$. For each $\{s_1, s_2, \ldots, s_l\} \in \mathcal{C}$, we increase $\mathcal{P}$ by

$(d \leftarrow s_1, s_2, \ldots, s_l)[1,1]$. Now, it is easy to see that No is the correct answer for $\exists(d)[\frac{1}{2},1]$ to $\mathcal{P}$ iff S can be partitioned into two subsets S_1 and S_2 such that no member of $\mathcal{C}$ is entirely contained in S_1 or S_2.

To show NP-hardness for $n = k+1$ with $k \geq 3$, we give a polynomial reduction from the NP-complete problem of graph k-colorability [17]. Let (V,E) be a finite undirected graph. Let Φ be the least set that contains the 0-ary predicate symbols d and v for each node $v \in V$. We now construct a ground n-valued logic program $\mathcal{P}$ as follows. We start by initializing $\mathcal{P}$ with $\emptyset$. For each node $v \in V$, we increase $\mathcal{P}$ by $(v \leftarrow)[\frac{1}{k},1]$. For each edge $\{u,v\} \in E$, we increase $\mathcal{P}$ by $(d \leftarrow u,v)[1,1]$. Now, it is easy to see that No is the correct answer for $\exists(d)[\frac{1}{k},1]$ to $\mathcal{P}$ iff (V,E) is k-colorable. $\Box$

Theorem 2.4 *Let Φ be function-free. Let $\mathcal{P}$ be a fixed n-valued logic program and let $\mathcal{F}$ be a varying finite set of ground logical facts. Let $\mathcal{P} \cup \mathcal{F}$ contain all constant symbols from Φ. The problem of deciding whether* Yes *is the correct answer for a ground n-valued query $\exists(A_1,\ldots,A_l)[c,1]$ to $\mathcal{P} \cup \mathcal{F}$ is co-NP-complete.*

Proof sketch. We show that the problem of deciding whether No is the correct answer for $\exists(A_1,\ldots,A_l)[c,1]$ to $\mathcal{P} \cup \mathcal{F}$ is NP-complete.

The problem is in NP, by Theorem 2.3, since the number of all ground instances of n-valued program clauses in $\mathcal{P} \cup \mathcal{F}$ (which are built with all constant symbols from $\mathcal{P} \cup \mathcal{F}$) is polynomial in the input size of $\mathcal{F}$.

To show NP-hardness for $n = 3$, we give again a polynomial reduction from the NP-complete problem of hypergraph 2-colorability. The main idea is that $\mathcal{F}$ can be used to encode any set $\mathcal{C}$ of nonempty subsets of a finite set S: Let Φ contain exactly the constant symbols ω, c for each $c \in S$, and C for each $C \in \mathcal{C}$, the 0-ary predicate symbol d, the 1-ary predicate symbols s and s', the 2-ary predicate symbols h and h', and the 3-ary predicate symbol o. Let $\mathcal{X}$ contain the three object variables x, y, and z. Let $\mathcal{P}$ contain exactly $(s(y) \leftarrow s'(y))[\frac{1}{2},1]$, $(d \leftarrow h(x,y), h'(x,y))[1,1]$, and $(h(x,y) \leftarrow o(x,y,z), s(y), h(x,z))[1,1]$. We now construct $\mathcal{F}$ as follows. We start by initializing $\mathcal{F}$ with $\emptyset$. For each $c \in S$, we increase $\mathcal{F}$ by $(s'(c) \leftarrow)[1,1]$. For each $C \in \mathcal{C}$, we increase $\mathcal{F}$ by $(h(C,\omega) \leftarrow)[1,1]$, $(h'(C,c_1) \leftarrow)[1,1]$, $(o(C,c_{m_C},\omega) \leftarrow)[1,1]$, and all $(o(C,c_i,c_{i+1}) \leftarrow)[1,1]$ with $i \in [1\!:\!m_C - 1]$, where $(c_1,c_2,\ldots,c_{m_C})$ is an ordering of C. Now, we can show that No is the correct answer for $\exists(d)[\frac{1}{2},1]$ to $\mathcal{P} \cup \mathcal{F}$ iff S can be partitioned into S_1 and S_2 such that no member of $\mathcal{C}$ is entirely contained in S_1 or S_2.

The proof of NP-hardness for $n = k+1$ with $k \geq 3$ can again be done by a polynomial reduction from the NP-complete problem of graph k-colorability. Also here, $\mathcal{F}$ can be used to encode any finite undirected graph (V,E). $\Box$

Thus, probabilistic many-valued logic programming is computationally more complex than classical logic programming. More precisely, restricted deduction problems that are computationally tractable for classical logic programs are presumably intractable for many-valued logic programs. Hence, any attempt towards efficient probabilistic many-valued logic programming should be guided by looking for efficient special-case, average-case, or approximation techniques.

3 Many-Valued Logic Programming in $\mathrm{Pr}_n^\star$

In this section, we present a truth-functional approach to many-valued logic programming that approximates our probabilistic one. In particular, we introduce $\mathrm{Pr}_n^\star$-interpretations and compare them with L_n-interpretations. Moreover, we define many-valued logic programs in $\mathrm{Pr}_n^\star$ and discuss their model semantics, their fixpoint semantics, and especially their proof theory. Finally, we focus on the computational complexity of many-valued logic programming in $\mathrm{Pr}_n^\star$.

3.1 $\mathrm{Pr}_n^\star$-Interpretations

Probabilistic many-valued logic programming as introduced in Section 2.2 has a well-defined probabilistic semantics. However, its increased computational complexity compared to classical logic programming is quite discouraging for a broad use in practice, especially for a possible application in large knowledge-base and deductive database systems.

This increase in computational complexity seems to be mainly due to the probabilistic semantics in its full generality. In fact, we now provide a truth-functional approach to many-valued logic programming that approximates our probabilistic one and that is less computationally complex. The main idea is to restrict our attention to a certain kind of Pr_n-interpretations:

A $\mathrm{Pr}_n^\star$-*interpretation* is a Pr_n-interpretation $Pr = (D, \mathcal{I}, \mu)$ that satisfies

$$Pr_\sigma(A \wedge B) = \min(Pr_\sigma(A), Pr_\sigma(B)) \tag{2}$$

for all variable assignments σ and all atomic formulas A and B.

Interestingly, the property (2) can equivalently be expressed as follows.

Theorem 3.1 *Let $Pr = (D, \mathcal{I}, \mu)$ be a Pr_n-interpretation.*

It holds $Pr_\sigma(A \wedge B) = \min(Pr_\sigma(A), Pr_\sigma(B))$ for all variable assignments σ and all atomic formulas A and B iff all the interpretations $I \in \mathcal{I}$ with $\mu(I) > 0$ can be written in a sequence $(D, \pi_1), \ldots, (D, \pi_k)$ such that:

$$\pi_1(p) \supseteq \pi_2(p) \supseteq \cdots \supseteq \pi_k(p) \text{ for all predicate symbols } p \text{ from } \Phi. \tag{3}$$

Proof. The proof can be done like the proof of Theorem 3.1 in [27]. □

Let us now adapt the notions of models, satisfiability, logical consequence, and tight logical consequence to $\mathrm{Pr}_n^\star$-interpretations:

A $\mathrm{Pr}_n^\star$-*model* of a set of n-valued formulas $\mathcal{P}$ is a $\mathrm{Pr}_n^\star$-interpretation that is a model of $\mathcal{P}$. The set of n-valued formulas $\mathcal{P}$ is *satisfiable in* $\mathrm{Pr}_n^\star$ iff a $\mathrm{Pr}_n^\star$-model of $\mathcal{P}$ exists. The n-valued formula P is a *logical consequence in* $\mathrm{Pr}_n^\star$ of $\mathcal{P}$ iff each $\mathrm{Pr}_n^\star$-model of $\mathcal{P}$ is also a model of P. The n-valued formula $\mathsf{tv}(F) \geq c$ is a *tight logical consequence in* $\mathrm{Pr}_n^\star$ of $\mathcal{P}$ iff c is the minimum of all truth values $Pr_\sigma(F)$ in $\mathrm{Pr}_n^\star$-models Pr of $\mathcal{P}$ under variable assignments σ. In the sequel, to avoid confusions, we also add "in Pr_n" to the corresponding notions of Section 2.1.

The following theorem shows that tight logical consequences in $\mathrm{Pr}_n^\star$ provide approximations for logical and tight logical consequences in Pr_n. In particular, for many-valued logic programs $\mathcal{P}$ and formulas F, this theorem shows that $\mathcal{P} \models_{tight} \mathsf{tv}(F) \geq 0$ in $\mathrm{Pr}_n^\star$ immediately entails $\mathcal{P} \models_{tight} \mathsf{tv}(F) \geq 0$ in Pr_n.

Theorem 3.2 *Let $\mathcal{F}$ be a set of n-valued formulas, let F be a formula, and let c be a truth value from TV. If $\mathcal{F} \models_{tight} \mathsf{tv}(F) \geq c$ in $\mathrm{Pr}_n^\star$, then all truth values $d \in TV$ with $\mathcal{F} \models \mathsf{tv}(F) \geq d$ in Pr_n are contained in $\{0, \ldots, c\} \subseteq TV$.*

Proof. The claim follows immediately from the definition of tight logical consequence in $\mathrm{Pr}_n^\star$ and of logical consequence in Pr_n. □

3.2 Comparison with Ł$_n$-Interpretations

We now show that the truth value of classical program clauses in $\mathrm{Pr}_n^\star$-interpretations under variable assignments can be defined truth-functionally by the truth tables of the finite-valued Łukasiewicz logics Ł_n.

Let us first briefly describe how many-valued formulas are interpreted in Ł_n. For this purpose, we just have to define Ł_n-interpretations and the truth value of classical formulas in Ł_n-interpretations under variable assignments:

An Ł_n-*interpretation* $L = (D, \boldsymbol{\pi})$ consists of a nonempty domain D and a mapping $\boldsymbol{\pi}$ that assigns to each k-ary function symbol from Φ a mapping from D^k to D and to each k-ary predicate symbol from Φ a mapping from D^k to the set of truth values TV. The *truth value* $L_\sigma(F)$ of a formula F in the Ł_n-interpretation L under a variable assignment σ is inductively defined by:

- $L_\sigma(p(t_1, \ldots, t_k)) = \boldsymbol{\pi}(p)(\sigma(t_1), \ldots, \sigma(t_k))$,
- $L_\sigma(\neg F) = 1 - L_\sigma(F)$ and $L_\sigma(F \wedge G) = \min(L_\sigma(F), L_\sigma(G))$,
- $L_\sigma(F \vee G) = \max(L_\sigma(F), L_\sigma(G))$,
- $L_\sigma(F \leftarrow G) = \min(1, L_\sigma(F) - L_\sigma(G) + 1)$,
- $L_\sigma(\forall x\, F) = \min\{L_{\sigma[x/d]}(F) \mid d \in D\}$,
- $L_\sigma(\exists x\, F) = \max\{L_{\sigma[x/d]}(F) \mid d \in D\}$.

Let us now focus on the relationship between $\mathrm{Pr}_n^\star$- and Ł_n-interpretations. The next lemma shows that for logical combinations of certain formulas, the truth value in $\mathrm{Pr}_n^\star$-interpretations under variable assignments is defined like the truth value in Ł_n-interpretations under variable assignments.

Lemma 3.3 *Let $Pr = (D, \mathcal{I}, \mu)$ be a $\mathrm{Pr}_n^\star$-interpretation and let σ be a variable assignment. For all object variables $x \in \mathcal{X}$, all formulas F, and all formulas G and H that are built without the logical connectives $\neg$ and $\leftarrow$:*

$$Pr_\sigma(\neg F) = 1 - Pr_\sigma(F) \tag{4}$$
$$Pr_\sigma(G \wedge H) = \min(Pr_\sigma(G), Pr_\sigma(H)) \tag{5}$$
$$Pr_\sigma(G \vee H) = \max(Pr_\sigma(G), Pr_\sigma(H)) \tag{6}$$
$$Pr_\sigma(G \leftarrow H) = \min(1, Pr_\sigma(G) - Pr_\sigma(H) + 1) \tag{7}$$
$$Pr_\sigma(\forall x\, G) = \min\{Pr_{\sigma[x/d]}(G) \mid d \in D\} \tag{8}$$
$$Pr_\sigma(\exists x\, G) = \max\{Pr_{\sigma[x/d]}(G) \mid d \in D\}\,. \tag{9}$$

Proof. The proof can be done like the proof of Lemma 3.3 in [27]. □

This means that $\mathrm{Pr}_n^\star$- and Ł_n-interpretations give the same truth value to all formulas that are built without the logical connectives $\neg$ and $\leftarrow$, and to all logical combinations of these formulas (thus, also to classical program clauses):

Theorem 3.4 *Let $Pr = (D, \mathcal{I}, \mu)$ be a* $\mathrm{Pr}_n^\star$*-interpretation, let $L = (D, \boldsymbol{\pi})$ be an L_n-interpretation, and let σ be a variable assignment.*

If $Pr_\sigma(A) = L_\sigma(A)$ for all atomic formulas A, then $Pr_\sigma(G) = L_\sigma(G)$, $Pr_\sigma(\neg G) = L_\sigma(\neg G)$, and $Pr_\sigma(G \leftarrow H) = L_\sigma(G \leftarrow H)$ for all formulas G and H that are built without the logical connectives $\neg$ and $\leftarrow$.

Proof. The first claim holds by Lemma 3.3 (5), (6), (8), and (9): it can easily be proved by induction on the length of the formulas that are built without the logical connectives $\neg$ and $\leftarrow$. Thereafter, the second and the third claim follow immediately from Lemma 3.3 (4) and (7). □

Note that there also exist formulas with different truth values in $\mathrm{Pr}_n^\star$-interpretations and in L_n-interpretations:

Theorem 3.5 *There exist* $\mathrm{Pr}_n^\star$*-interpretations $Pr = (D, \mathcal{I}, \mu)$, L_n-interpretations $L = (D, \boldsymbol{\pi})$, variable assignments σ, and formulas G such that $Pr_\sigma(A) = L_\sigma(A)$ for all atomic formulas A and that $Pr_\sigma(G) \neq L_\sigma(G)$.*

Proof. We verify easily that if the vocabulary Φ contains at least one predicate symbol, then there exists a $\mathrm{Pr}_n^\star$-interpretation $Pr = (D, \mathcal{I}, \mu)$, an L_n-interpretation $L = (D, \boldsymbol{\pi})$, a variable assignments σ, and an atomic formula F such that $Pr_\sigma(A) = L_\sigma(A)$ for all atomic formulas A and that $Pr_\sigma(F) = L_\sigma(F) = \frac{1}{n-1}$.

Now, for $G = F \wedge \neg F$, we get immediately $Pr_\sigma(G) = Pr_\sigma(F \wedge \neg F) = 0$ and $L_\sigma(G) = L_\sigma(F \wedge \neg F)$ $= \min(L_\sigma(F), L_\sigma(\neg F)) = \min(\frac{1}{n-1}, \frac{n-2}{n-1}) = \frac{1}{n-1}$. □

This last theorem is not surprising, since $\mathrm{Pr}_n^\star$-interpretations are just special Pr_n-interpretations. That is, they still satisfy the axioms of probability. In particular, $\mathrm{Pr}_n^\star$-interpretations always give the same truth value to logically equivalent formulas. L_n-interpretations, in contrast, do not have this property.

3.3 Many-Valued Logic Programs

We keep the definitions of many-valued program clauses and many-valued programs from Section 2.2. In particular, the semantics of many-valued program clauses in $\mathrm{Pr}_n^\star$-interpretations is already given by the semantics of many-valued formulas in Pr_n-interpretations. The truth of many-valued program clauses in $\mathrm{Pr}_n^\star$-interpretations is then additionally characterized as follows.

Lemma 3.6 *For all* $\mathrm{Pr}_n^\star$*-interpretations $Pr = (D, \mathcal{I}, \mu)$, all variable assignments σ, and all n-valued program clauses $(H \leftarrow B_1, \ldots, B_k)[c, 1]$:*

$$(H \leftarrow B_1, \ldots, B_k)[c, 1] \textit{ is true in } Pr \textit{ under } \sigma \textit{ iff}$$
$$Pr_\sigma(H) \geq c - 1 + \min(Pr_\sigma(B_1), \ldots, Pr_\sigma(B_k))\,.$$

Proof. The proof can be done like the proof of Lemma 5.1 in [27]. □

We also keep the syntax of many-valued queries from Section 2.2. We just redefine correct answers, correct answer substitutions, and tight answer substitutions: Given an n-valued query $Q_c = \exists(A_1, \ldots, A_l)[c, 1]$ with $c \in TV$, we are interested in its *correct answer substitutions in* $\mathrm{Pr}_n^\star$, which are substitutions θ such that $\mathcal{P} \models \mathsf{tv}(A_1\theta \wedge \cdots \wedge A_l\theta) \geq c$ in $\mathrm{Pr}_n^\star$ and that θ acts only on variables in Q_c. The *correct answer in* $\mathrm{Pr}_n^\star$ for Q_c is Yes if a correct answer substitution in $\mathrm{Pr}_n^\star$ exists and No otherwise. Given an n-valued query $Q_x = \exists(A_1, \ldots, A_l)[x, 1]$ with $x \in \mathcal{X}$, we are interested in its *tight answer substitutions in* $\mathrm{Pr}_n^\star$, which are substitutions θ such that $\mathcal{P} \models_{tight} \mathsf{tv}(A_1\theta \wedge \cdots \wedge A_l\theta) \geq x\theta$ in $\mathrm{Pr}_n^\star$, that θ acts only on variables in Q_x, and that $x\theta$ is a truth value from TV.

In the sequel, we use *many-valued logic programming in* $\mathrm{Pr}_n^\star$ as a synonym for the problem of deciding whether Yes is the correct answer in $\mathrm{Pr}_n^\star$ for a given ground many-valued query to a many-valued logic program.

3.4 Example

Let $n = 101$ and let $\mathcal{P}$ be the n-valued logic program from Section 2.3. The correct answer in $\mathrm{Pr}_n^\star$ for the n-valued query $\exists(re(h, o))[.99, 1]$ to $\mathcal{P}$ is No, whereas the correct answer in $\mathrm{Pr}_n^\star$ for $\exists(re(h, U))[.8,1]$ to $\mathcal{P}$ is Yes (note that all the correct answer substitutions in $\mathrm{Pr}_n^\star$ for $\exists(re(h, U))[.8,1]$ to $\mathcal{P}$ are given by $\{U/a\}$, $\{U/b\}$, and $\{U/o\}$). Finally, the unique tight answer substitution in $\mathrm{Pr}_n^\star$ for the n-valued query $\exists(re(h, o))[X, 1]$ to $\mathcal{P}$ is given by $\{X/.8\}$.

3.5 Important Properties

We now show that the two nice properties of probabilistic many-valued logic programs discussed in Section 2.4 carry over to the truth-functional setting.

Theorem 3.7 *Every n-valued logic program is satisfiable in* $\mathrm{Pr}_n^\star$.

Proof. The claim follows from the proof of Theorem 2.1, since the Herbrand Pr_n-interpretation $(\mathcal{I}, \mu)$ used there is a $\mathrm{Pr}_n^\star$-interpretation. □

The next theorem justifies in particular the model and fixpoint semantics of the next subsection, which is restricted to Herbrand $\mathrm{Pr}_n^\star$-interpretations.

Theorem 3.8 *Let $\mathcal{P}$ be a set of quantifier-free n-valued formulas and let F be a ground formula. For each* $\mathrm{Pr}_n^\star$*-interpretation Pr with $Pr \models \mathcal{P}$, there exists a Herbrand* $\mathrm{Pr}_n^\star$*-interpretation Pr' with $Pr' \models \mathcal{P}$ and $Pr(F) = Pr'(F)$.*

Proof. The claim follows from the proof of Theorem 2.2, since for all $\mathrm{Pr}_n^\star$-interpretations $Pr = (D, \mathcal{I}, \mu)$, the Herbrand Pr_n-interpretation $Pr' = (\mathcal{I}', \mu')$ in the proof of Theorem 2.2 is a $\mathrm{Pr}_n^\star$-interpretation. □

3.6 Model and Fixpoint Semantics

We briefly discuss the model and fixpoint semantics of many-valued logic programs in $\mathrm{Pr}_n^\star$ [27]. In the sequel, let $\mathcal{P}$ be an n-valued logic program.

We focus on Herbrand $\mathrm{Pr}_n^\star$-interpretations, which we identify with fuzzy sets [40]. In detail, each Herbrand $\mathrm{Pr}_n^\star$-interpretation $(\mathcal{I}, \mu)$ is identified with the fuzzy set $\boldsymbol{I}\colon HB_\Phi \to TV$, where $\boldsymbol{I}[A]$, for all $A \in HB_\Phi$, is the sum of all $\mu(I)$ with $I \in \mathcal{I}$ and $I \models A$. We subsequently use bold symbols to denote such fuzzy sets. The fuzzy sets $\emptyset$ and $\boldsymbol{HB}_\Phi$ are defined by $\emptyset[A] = 0$ and $\boldsymbol{HB}_\Phi[A] = 1$ for all $A \in HB_\Phi$. Finally, we define the intersection, the union, and the subset relation for fuzzy sets $\boldsymbol{S}_1$

and $\boldsymbol{S}_2$ as usual by $\boldsymbol{S}_1 \cap \boldsymbol{S}_2 = \min(\boldsymbol{S}_1, \boldsymbol{S}_2)$, $\boldsymbol{S}_1 \cup \boldsymbol{S}_2 = \max(\boldsymbol{S}_1, \boldsymbol{S}_2)$, and $\boldsymbol{S}_1 \subseteq \boldsymbol{S}_2$ iff $\boldsymbol{S}_1 = \boldsymbol{S}_1 \cap \boldsymbol{S}_2$, respectively.

We define the immediate consequence operator $\boldsymbol{T}_{\mathcal{P}}$ on the set of all subsets of $\boldsymbol{HB}_\Phi$ as follows. For all $\boldsymbol{I} \subseteq \boldsymbol{HB}_\Phi$ and $H \in HB_\Phi$:

$$\begin{aligned} \boldsymbol{T}_{\mathcal{P}}(\boldsymbol{I})[H] \;=\; & \max(\{c-1+\min(\boldsymbol{I}[B_1], \ldots, \boldsymbol{I}[B_k]) \mid (H \leftarrow B_1, \ldots, B_k)[c,1] \text{ is a} \\ & \text{ground instance of an } n\text{-valued program clause in } \mathcal{P}\} \cup \{0\}) \,. \end{aligned}$$

Note that we canonically define $\min(\boldsymbol{I}[B_1], \ldots, \boldsymbol{I}[B_k]) = 1$ for $k = 0$.

For all $\boldsymbol{I} \subseteq \boldsymbol{HB}_\Phi$, we define $\boldsymbol{T}_{\mathcal{P}}\!\uparrow\omega(\boldsymbol{I})$ as the union of all $\boldsymbol{T}_{\mathcal{P}}\!\uparrow l(\boldsymbol{I})$ with $l < \omega$, where $\boldsymbol{T}_{\mathcal{P}}\!\uparrow 0(\boldsymbol{I}) = \boldsymbol{I}$ and $\boldsymbol{T}_{\mathcal{P}}\!\uparrow(l+1)(\boldsymbol{I}) = \boldsymbol{T}_{\mathcal{P}}(\boldsymbol{T}_{\mathcal{P}}\!\uparrow l(\boldsymbol{I}))$ for all $l < \omega$. Finally, we adopt the usual convention to abbreviate $\boldsymbol{T}_{\mathcal{P}}\!\uparrow\alpha(\emptyset)$ by $\boldsymbol{T}_{\mathcal{P}}\!\uparrow\alpha$.

The model and fixpoint semantics of many-valued logic programs in $\mathrm{Pr}_n^\star$ is now expressed by the following important theorem:

Theorem 3.9 $\bigcap\{\boldsymbol{I} \mid \boldsymbol{I} \subseteq \boldsymbol{HB}_\Phi,\, \boldsymbol{I} \models \mathcal{P}\} \;=\; \mathit{lfp}(\boldsymbol{T}_{\mathcal{P}}) \;=\; \boldsymbol{T}_{\mathcal{P}}\!\uparrow\omega\,.$

Proof. The claim is proved in [27]. □

This model and fixpoint semantics yields the following characterization of tight answer substitutions for object-ground many-valued queries:

Theorem 3.10 *Let $\mathcal{P}$ be an n-valued logic program and let $\exists(A_1, \ldots, A_l)[x,1]$ be an object-ground n-valued query with $x \in \mathcal{X}$.*

The tight answer substitution in $\mathrm{Pr}_n^\star$ for $\exists(A_1, \ldots, A_l)[x,1]$ to $\mathcal{P}$ is given by $\{x/c\}$, where c is the minimum of all $\boldsymbol{T}_{\mathcal{P}}\!\uparrow\omega[A_i]$ with $i \in [1\!:\!l]$.

Proof. By Lemma 3.3 (5), $Pr(A_1 \wedge \cdots \wedge A_l) = \min(Pr(A_1), \ldots, Pr(A_l))$ for all $\mathrm{Pr}_n^\star$-interpretations Pr. Thus, the claim follows from Theorems 3.8 and 3.9. □

3.7 Proof Theory

We now present $\mathrm{SLDPr}_n^\star$-resolution for many-valued logic programs in $\mathrm{Pr}_n^\star$ and prove its soundness and completeness. The $\mathrm{SLDPr}_n^\star$-resolution is an extension of the classical SLD-resolution (see, for example, [1]).

We subsequently try to stay as close as possible to the standard notions of classical SLD-resolution. The only nonstandard notion that we introduce is the one of resolution strings, which replace negative program clauses. In the sequel, we abbreviate many-valued facts $(A \leftarrow\ )[c,1]$ by $(A)[c,1]$.

A *resolution string* is a finite list $(A_1)[a_1,1] \ldots (A_m)[a_m,1]$ of n-valued facts $(A_1)[a_1,1], \ldots, (A_m)[a_m,1]$ with $a_1, a_2, \ldots, a_m > 0$ and $m \geq 0$.

A substitution θ is applied to a resolution string by replacing each contained atomic formula A_i by $A_i\theta$. For n-valued program clauses P_1 and P_2, we say P_1 is a *variant* of P_2 iff P_1 is an instance of P_2 and P_2 is an instance of P_1. The notions of unifiers and most general unifiers (mgu) are defined as usual.

The resolution string $(\alpha(B_1)[b,1] \ldots (B_k)[b,1]\,\omega)\theta$ is a *resolvent* of the resolution string $\alpha(A)[a,1]\,\omega$ and the n-valued program clause $(H \leftarrow B_1, \ldots, B_k)[c,1]$ *with* mgu θ iff A and H unify with mgu θ, $a \leq c$, and $b = a - c + 1$.

Note that, for resolution strings $\alpha(A)[a,1]\omega$ and n-valued program clauses $(H \leftarrow B_1, \ldots, B_k)[c,1]$, the resolvent $(\alpha(B_1)[b,1] \ldots (B_k)[b,1]\omega)\theta$ is a resolution string, since $0 < a \leq c \leq 1$ and $b = a - c + 1$ entails $0 < b \leq 1$.

An $\mathrm{SLDPr}_n^\star$-*derivation* of a resolution string R_0 from an n-valued logic program $\mathcal{P}$ is a maximal sequence $R_0, (C_0, \theta_0), R_1, (C_1, \theta_1), \ldots,$ where $R_0, R_1, \ldots$ is a sequence of resolution strings, $C_0, C_1, \ldots$ is a sequence of variants of clauses from $\mathcal{P}$, and $\theta_0, \theta_1, \ldots$ is a sequence of substitutions such that R_{i+1} is a resolvent of R_i and C_i with mgu θ_i and such that C_i does not have any variables in common with $R_0, C_0, \ldots, R_{i-1}$. If a resolution string R_j is empty, then it is the last one in a derivation. Such an $\mathrm{SLDPr}_n^\star$-derivation is called *successful*.

Next, we show that $\mathrm{SLDPr}_n^\star$-resolution is a sound and complete technique for correct query answering in $\mathrm{Pr}_n^\star$. That is, we now prove that for n-valued logic programs $\mathcal{P}$ and n-valued queries $Q_c = \exists(A_1, \ldots, A_l)[c,1]$ with $c > 0$, the correct answer in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$ is Yes iff a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c,1] \ldots (A_l)[c,1]$ from $\mathcal{P}$ exists. We also show that each successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c,1] \ldots (A_l)[c,1]$ from $\mathcal{P}$ with the sequence of substitutions $\theta_0, \theta_1, \ldots, \theta_j$ provides a correct answer substitution in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$ by the substitution $\theta_0\theta_1 \ldots \theta_j$ restricted to the variables in Q_c.

To prove the soundness of $\mathrm{SLDPr}_n^\star$-resolution, we need some preparation:

Lemma 3.11 *If η is a substitution and $(\alpha(B_1)[b,1] \ldots (B_k)[b,1]\omega)\theta$ is a resolvent of $\alpha(A)[a,1]\,\omega$ and $(H \leftarrow B_1, \ldots, B_k)[c,1]$ with mgu θ, then $(A\theta\eta)[a,1]$ is logically entailed by $\{(H \leftarrow B_1, \ldots, B_k)[c,1],$ $(B_1\theta\eta)[b,1], \ldots, (B_k\theta\eta)[b,1]\}$ in $\mathrm{Pr}_n^\star$.*

Proof. For $k = 0$, the n-valued fact $(A\theta\eta)[a,1]$ is clearly a logical consequence of $\{(H)[c,1]\}$ with $a \leq c$. To show the claim for $k \geq 1$, let $Pr = (D, \mathcal{I}, \mu)$ be a $\mathrm{Pr}_n^\star$-interpretation with $Pr \models_\sigma (H \leftarrow B_1, \ldots, B_k)[c,1]$ and $Pr \models_\sigma (B_i\theta\eta)[b,1]$ for all $i \in [1\!:\!k]$ and all variable assignments σ. By Lemma 3.6, we then get $Pr_\sigma(H) \geq c - 1 + \min(Pr_\sigma(B_1), \ldots, Pr_\sigma(B_k))$ and $Pr_\sigma(B_i\theta\eta) \geq b$ for all $i \in [1\!:\!k]$ and all variable assignments σ. Hence, since $b = a - c + 1$, we get $Pr_\sigma(A\theta\eta) = Pr_\sigma(H\theta\eta) \geq a$ for all variable assignments σ. $\square$

We are now ready to show the soundness of $\mathrm{SLDPr}_n^\star$-resolution:

Theorem 3.12 *Let $\mathcal{P}$ be an n-valued logic program and $Q_c = \exists(A_1, \ldots, A_l)[c,1]$ be an n-valued query with $c > 0$. If there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c,1] \ldots (A_l)[c,1]$ from $\mathcal{P}$ with the sequence of substitutions $\theta_0, \theta_1, \ldots, \theta_j$, then the substitution $\theta_0\theta_1 \ldots \theta_j$ restricted to the variables in Q_c is a correct answer substitution in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$.*

Proof. Let us first give a preparative definition: for a resolution string R, we use $\mathcal{F}(R)$ to denote the set of all n-valued facts that occur in R.

Let $R_0, (C_0, \theta_0), R_1, (C_1, \theta_1), \ldots, R_{j+1}$ be a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c,1] \ldots (A_l)[c,1]$ from $\mathcal{P}$. By Lemma 3.11, $\mathcal{F}(R_{i+1}\eta) \cup \{C_i\} \models \mathcal{F}(R_i\theta_i\eta)$ for all $i \in [0\!:\!j]$ and all substitutions η. Hence, by induction on i, one can easily show that $\mathcal{P} \models \mathcal{F}(R_i\theta_i\theta_{i+1} \ldots \theta_j)$ for all $i \in [0:j]$. In particular, we also get $\mathcal{P} \models \mathcal{F}(R_0\theta_0\theta_1 \ldots \theta_j)$. That is, if the $\mathrm{Pr}_n^\star$-interpretation Pr is a model of $\mathcal{P}$, then $Pr_\sigma(A_i\theta_0\theta_1 \ldots \theta_j) \geq c$ for all $i \in [1:l]$ and all variable assignments σ. Thus, by Lemma 3.3 (5), $Pr_\sigma((A_1 \wedge \cdots \wedge A_l)\theta_0\theta_1 \ldots \theta_j) \geq c$ for all variable assignments σ. Hence, we get $\mathcal{P} \models \mathsf{tv}((A_1 \wedge \cdots \wedge A_l)\theta_0\theta_1 \ldots \theta_j) \geq c$. $\square$

The next theorem shows the completeness of $\mathrm{SLDPr}_n^\star$-resolution.

Theorem 3.13 *Let $\mathcal{P}$ be an n-valued logic program and $Q_c = \exists(A_1, \ldots, A_l)[c,1]$ be an n-valued query with $c > 0$. If* Yes *is the correct answer in* $\mathrm{Pr}_n^\star$ *for Q_c to $\mathcal{P}$, then a successful* $\mathrm{SLDPr}_n^\star$*-derivation of $(A_1)[c,1]\ldots(A_l)[c,1]$ from $\mathcal{P}$ exists.*

Proof. The argumentation follows similar lines to the proof of completeness of classical SLD-resolution (see, for example, [1]).

First, note that for all substitutions η, the existence of a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1\eta)[c,1]\ldots(A_l\eta)[c,1]$ from $\mathcal{P}$ entails the existence of a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c,1]$ $\ldots(A_l)[c,1]$ from $\mathcal{P}$. This result can be proved in the same way like its classical counterpart (see, for example, [1]).

Next, we show that for all $A \in HB_\Phi$ with $\boldsymbol{T}_\mathcal{P}\!\uparrow\omega[A] > 0$, there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A)[\boldsymbol{T}_\mathcal{P}\!\uparrow\omega[A],1]$ from $\mathcal{P}$. Since $\boldsymbol{T}_\mathcal{P}\!\uparrow\omega[A] > 0$, there exists an $m > 0$ with $\boldsymbol{T}_\mathcal{P}\!\uparrow m[A] = \boldsymbol{T}_\mathcal{P}\!\uparrow\omega[A]$. We now show by induction on m that a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A)[\boldsymbol{T}_\mathcal{P}\!\uparrow m[A],1]$ from $\mathcal{P}$ exists: For $m = 1$, the claim obviously holds. Let us now consider the case $m > 1$. There exists a ground instance $(H\eta\leftarrow B_1\eta, \ldots, B_k\eta)[c,1]$ of an n-valued program clause $(H\leftarrow B_1, \ldots, B_k)[c,1]$ from $\mathcal{P}$ with $A{=}H\eta$ and $\boldsymbol{T}_\mathcal{P}\!\uparrow m[A] = c{-}1{+}b$ where $b = \min(\boldsymbol{T}_\mathcal{P}\!\uparrow(m{-}1)[B_1\eta], \ldots, \boldsymbol{T}_\mathcal{P}\!\uparrow(m{-}1)[B_k\eta])$. By the induction hypothesis, there exist successful $\mathrm{SLDPr}_n^\star$-derivations of $(B_i\eta)[\boldsymbol{T}_\mathcal{P}\!\uparrow(m{-}1)[B_i\eta],1]$ from $\mathcal{P}$ for all $i \in [1:k]$. Now, it is easy to see that then there also exist successful $\mathrm{SLDPr}_n^\star$-derivations of $(B_i\eta)[b,1]$ from $\mathcal{P}$ for all $i \in [1:k]$. Hence, since all $B_i\eta$ with $i \in [1:k]$ are ground, there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(B_1\eta)[b,1]\ldots(B_k\eta)[b,1]$ from $\mathcal{P}$. If A and H now unify with mgu θ, then $B_i\eta, \ldots, B_k\eta$ are ground instances of $B_i\theta, \ldots, B_k\theta$. Hence, there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(B_1\theta)[b,1]\ldots(B_k\theta)[b,1]$ from $\mathcal{P}$. Thus, since $(B_1\theta)[b,1]\ldots(B_k\theta)[b,1]$ is the resolvent of $(A)[\boldsymbol{T}_\mathcal{P}\!\uparrow m[A],1]$ and the n-valued program clause $(H\leftarrow B_1, \ldots, B_k)[c,1]$ from $\mathcal{P}$ with mgu θ, there also exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A)[\boldsymbol{T}_\mathcal{P}\!\uparrow m[A],1]$ from $\mathcal{P}$.

Since Yes is the correct answer in $\mathrm{Pr}_n^\star$ for $\exists(A_1, \ldots, A_l)[c,1]$ to $\mathcal{P}$, there exists a substitution θ such that $A_1\theta, \ldots, A_l\theta \in HB_\Phi$ and that Yes is the correct answer in $\mathrm{Pr}_n^\star$ for $\exists(A_1\theta, \ldots, A_l\theta)[c,1]$ to $\mathcal{P}$. Hence, we get $\boldsymbol{T}_\mathcal{P}\!\uparrow\omega[A_i\theta] \geq c$ for all $i \in [1:l]$. Furthermore, there exist successful $\mathrm{SLDPr}_n^\star$-derivations of $(A_i\theta)[\boldsymbol{T}_\mathcal{P}\!\uparrow\omega[A_i\theta],1]$ from $\mathcal{P}$ for all $i \in [1:l]$. Now, it is easy to see that then there also exist successful $\mathrm{SLDPr}_n^\star$-derivations of $(A_i\theta)[c,1]$ from $\mathcal{P}$ for all $i \in [1:l]$. Hence, since all $A_i\theta$ with $i \in [1:l]$ are ground, there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1\theta)[c,1]\ldots(A_l\theta)[c,1]$ from $\mathcal{P}$, and thus there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c,1]\ldots(A_l)[c,1]$ from $\mathcal{P}$. $\Box$

Finally, by following the same argumentation like in classical logic programming, the completeness of $\mathrm{SLDPr}_n^\star$-resolution can be extended as follows:

Theorem 3.14 *Let $\mathcal{P}$ be an n-valued logic program and $Q_c = \exists(A_1, \ldots, A_l)[c,1]$ be an n-valued query with $c > 0$. For each correct answer substitution η in* $\mathrm{Pr}_n^\star$ *for Q_c to $\mathcal{P}$, there exists a successful* $\mathrm{SLDPr}_n^\star$*-derivation of $(A_1)[c,1]\ldots(A_l)[c,1]$ from $\mathcal{P}$ with the sequence of substitutions $\theta_0, \theta_1, \ldots, \theta_j$ such that $(A_1 \wedge \ldots \wedge A_l)\eta$ is an instance of $(A_1 \wedge \ldots \wedge A_l)\theta_0\theta_1\ldots\theta_j$.*

Proof. The claim can be proved in the same way like its counterpart from classical logic programming (see, for example, [1]). $\Box$

3.8 Computational Complexity

Also many-valued logic programming in $\mathrm{Pr}_n^\star$ in its full generality is undecidable, since it is a generalization of classical logic programming.

Let us now focus on the computational complexity of the two decidable special cases that generalize propositional logic programming and the decision problem that defines the data complexity of datalog. Crucially, in contrast to the probabilistic many-valued generalizations of Sections 2.5, the truth-functional ones are P-complete. This follows from the next two theorems, which even show that the optimization problem of computing tight answer substitutions for object-ground many-valued queries is P-complete in the two special cases.

Theorem 3.15 *The optimization problem of computing the tight answer substitution in* $\mathrm{Pr}_n^\star$ *for an object-ground* n*-valued query* $\exists(A_1, \ldots, A_l)[x, 1]$*, with* $x \in \mathcal{X}$*, to a ground* n*-valued logic program* $\mathcal{P}$ *is P-complete.*

Proof. The claim holds by Theorem 3.10 and the proof of Theorem 5.5 in [27]. □

Theorem 3.16 *Let* Φ *be function-free. Let* $\mathcal{P}$ *be a fixed* n*-valued logic program, let* $\mathcal{F}$ *be a varying finite set of ground* n*-valued facts. Let* $\mathcal{P} \cup \mathcal{F}$ *contain all constant symbols from* Φ*. The optimization problem of computing the tight answer substitution in* $\mathrm{Pr}_n^\star$ *for an object-ground* n*-valued query* $\exists(A_1, \ldots, A_l)[x, 1]$*, with* $x \in \mathcal{X}$*, to* $\mathcal{P} \cup \mathcal{F}$ *is P-complete.*

Proof. The claim holds by Theorem 3.10 and the proof of Theorem 5.6 in [27]. □

4 Summary and Outlook

We introduced probabilistic many-valued logic programs in which the implication connective is interpreted as material implication. We showed that probabilistic many-valued logic programming is computationally more complex than classical logic programming. We then focused on the approximation of probabilistic many-valued logic programming by many-valued logic programming in $\mathrm{Pr}_n^\star$. In particular, we introduced a proof theory for many-valued logic programming in $\mathrm{Pr}_n^\star$ and showed its soundness and completeness.

Crucially, many-valued logic programs in $\mathrm{Pr}_n^\star$ have both a probabilistic semantics in probabilities over a set of possible worlds and a truth-functional semantics in the finite-valued Łukasiewicz logics $\mathrm{Ł}_n$. Moreover, many-valued logic programming in $\mathrm{Pr}_n^\star$ has a model and fixpoint characterization, a proof theory, and computational properties that are very similar to those of classical logic programming. Hence, it is very worth being studied more deeply.

In particular, a very interesting topic of future research is to explore many-valued logic programming in $\mathrm{Pr}_n^\star$ with (classical and nonmonotonic) negation in rule bodies and disjunction in rule heads.

Finally, this paper showed how presumably intractable probabilistic deduction problems in artificial intelligence can be tackled by efficient approximation techniques based on truth-functional many-valued logics.

References

[1] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 10, pages 493–574. MIT Press, 1990.

[2] M. Baaz and C. G. Fermüller. Resolution-based theorem proving for many-valued logics. *Journal of Symbolic Computation*, 19(4):353–391, 1995.

[3] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge: A Logical Approach to Probabilities.* MIT Press, Cambridge, USA, 1990.

[4] F. Bacchus, A. Grove, J. Y. Halpern, and D. Koller. From statistical knowledge bases to degrees of beliefs. *Artificial Intelligence*, 87:75–143, 1996.

[5] J. F. Baldwin. Evidential support logic programming. *Fuzzy Sets and Systems*, 24:1–26, 1987.

[6] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.

[7] R. Carnap. *Logical Foundations of Probability.* University of Chicago Press, Chicago, 1950.

[8] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 82–101, Ulm, Germany, 1997.

[9] B. de Finetti. *Theory of Probability.* Wiley, New York, 1974.

[10] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. In *Proceedings of the 14th International Conference on Logic Programming*, pages 391–405, 1997.

[11] J. P. Delahaye and V. Thibau. Programming in three-valued logic. *Theoretical Computer Science*, 78:189–216, 1991.

[12] G. Escalada-Imaz and F. Manyà. Efficient interpretation of propositional multi-valued logic programs. In *Proceedings 5th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '94)*, volume 945 of *LNCS*, pages 428–239. Springer, 1995.

[13] R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87:78–128, 1990.

[14] M. Fitting. Partial models and logic programming. *Theoretical Computer Science*, 48:229–255, 1986.

[15] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(1–2):91–116, 1991.

[16] B. R. Gaines. Fuzzy and probability uncertainty logics. *Information and Control*, 38:154–169, 1978.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[18] R. Hähnle. Exploiting data dependencies in many-valued logics. *Journal of Applied Non-Classical Logics*, 6(1):49–69, 1996.

[19] R. Hähnle and G. Escalada-Imaz. Deduction in many-valued logics: a survey. *Mathware & Soft Computing*, IV(2):69–97, 1997.

[20] J. Y. Halpern. An analysis of first-order logics of probability. *Artif. Intell.*, 46:311–350, 1990.

[21] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(3–4):335–367, 1992.

[22] L. V. S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Proceedings of the International Logic Programming Symposium*, pages 254–268, 1994.

[23] J.-L. Lassez and M. J. Maher. Optimal fixedpoints of logic programs. *Theoretical Computer Science*, 39:15–25, 1985.

[24] S. M. Leach and J. J. Lu. Query processing in annotated logic programming: Theory and implementation. *Journal of Intelligent Information Systems*, 6(1):33–58, 1996.

[25] J. W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 2nd ed., 1987.

[26] J. J. Lu. Logic programming with signs and annotations. *Journal of Logic and Computation*, 6(6):755–778, 1996.

[27] T. Lukasiewicz. Many-valued first-order logics with probabilistic semantics. Presented at the *Annual Conference of the European Association for Computer Science Logic*, Brno, Czech Republic, 1998.

[28] T. Lukasiewicz. Magic inference rules for probabilistic deduction under taxonomic knowledge. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 354–361. Morgan Kaufmann Publishers, 1998.

[29] T. Lukasiewicz. Probabilistic deduction with conditional constraints over basic events. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference*, pages 380–391. Morgan Kaufmann Publishers, 1998.

[30] T. Lukasiewicz. Probabilistic logic programming. In *Proceedings of the 13th Biennial European Conference on Artificial Intelligence*, pages 388–392. J. Wiley & Sons, 1998.

[31] D. Mundici and N. Olivetti. Resolution and model building in the infinite-valued calculus of Łukasiewicz. *Theoretical Computer Science*, 200(1-2):335–366, 1998.

[32] R. T. Ng. Semantics, consistency, and query processing of empirical deductive databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):32–49, 1997.

[33] R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101:150–201, 1992.

[34] R. T. Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning*, 10(2):191–235, 1993.

[35] R. T. Ng and V. S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Information and Computation*, 110:42–83, 1994.

[36] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–88, 1986.

[37] N. Rescher. *Many-valued Logic.* McGraw-Hill, New York, 1969.

[38] J. B. Rosser and A. R. Turquette. *Many-valued Logics.* North-Holland, Amsterdam, 1952.

[39] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

[40] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.