

ProbLog Technology for Inference in a Probabilistic First Order Logic

Maurice Bruynooghe and Theofrastos Mantadelis and Angelika Kimmig and Bernd Gutmann
and Joost Vennekens and Gerda Janssens and Luc De Raedt¹

Abstract. We introduce First Order ProbLog, an extension of first order logic with soft constraints where formulas are guarded by probabilistic facts. The paper defines a semantics for FOProbLog, develops a translation into ProbLog, a system that allows a user to compute the probability of a query in a similar setting restricted to Horn clauses, and reports on initial experience with inference.

1 Introduction

A problem for languages that combine probability theory with *expressive* logical formulas is that probabilistic and logical knowledge might interact in complex ways, leading to a semantics that is too complicated to understand, and inference/learning that is too slow to be of practical use. Initial research into Probabilistic Logic Learning therefore mainly focused on adding probabilities to restricted logical languages, such as definite clause logic [3, 13]. Since then, the trend has been to extend the expressivity of the logical language, e.g., to normal clauses [10]. In the past few years there has been a lot of attention to Markov Logic [12], a probabilistic logic used in statistical relational learning [1]. Markov Logic consists of a set of weighted logical formulas each of which can be regarded as a soft constraint. If such a soft constraint is violated by a possible world, the probability of the world does not become zero, as in first order logic, but gracefully decreases. The higher the weight, the harder the constraint becomes, with infinite weights corresponding to the usual logical interpretation. While Markov Logic has been very successful as a framework for statistical relational learning and has been applied in several challenging applications, such as natural language processing and entity resolution, Markov Logic models are hard to interpret and not really suitable as a knowledge representation tool. The reason is that the weights cannot directly be interpreted as probabilities and also that the probability of a formula depends non-linearly on all weights in the theory.

This paper contributes a new formalism, called First Order ProbLog, using the Markov Logic idea of soft constraints, but in which each first order formula is annotated with the probability that a grounding of the formula – independently of anything else – holds. This gives a logic very similar to that of Nilson [9]. The questions arise whether inference is feasible and how one can cope with inconsistency. These questions are explored in this paper. Using ideas of Stickel [15], we translate a theory in our logic into a ProbLog program [5] that also includes clauses for inferring inconsistency (with head *false*). We show how to use the ProbLog machinery to assign a minimal and a maximal probability to the truth of the query while taking consistency requirements into account.

2 First Order ProbLog and its Semantics

A ProbLog program [4, 5] consists of a set of facts annotated with probabilities – called *probabilistic facts* – together with standard definite clauses that can have positive and negative probabilistic facts in their body. The semantics is defined through belief sets which correspond to least Herbrand models of the clauses together with subsets of the probabilistic facts. If fact f_i is annotated with p_i , f_i is included in a belief set with probability p_i and left out with probability $1 - p_i$. The different facts are assumed to be probabilistically independent, however, negative probabilistic facts in clause bodies allow the user to enforce a choice between two clauses.

In this paper we define FOProbLog, a language similar to ProbLog, but using full first order logic formulas instead of definite clauses. A FOProbLog statement is of the form $\forall \bar{x}. \Psi_1 : \alpha_1 \vee \dots \vee \Psi_n : \alpha_n$ where the α_i are non-zero probabilities with sum 1 and the Ψ_i are first order formulas with free variables included in \bar{x} . If Ψ_i is *true*, it can be omitted (in which case the sum of the probabilities becomes smaller than 1); if $\alpha_1 = 1$, it can be omitted as well. These formulas express independent beliefs about the world. Each belief is disjunctive: we believe that for each grounding of \bar{x} exactly one of a number of possibilities holds, but we don't know which one, so we attach a probability to each of the disjuncts. We can now have a number of different "complete belief sets", which are formed by believing precisely one disjunct from every grounded disjunction according to its probability.

Ex. 1 *Our running example uses a domain with a single constant Floris and theories consisting of subsets of the following formulas*

- (1) $male(Floris) : 0.4 \vee female(Floris) : 0.6$
- (2) $\forall x.(cs(x) \rightarrow male(x) : 0.8) \vee (cs(x) \rightarrow female(x) : 0.2)$
- (3) $\forall x.\neg(male(x) \wedge female(x))$
- (4) $cs(Floris)$

If the theory contains only formula (1), expressing a prior belief about the name Floris being male or female, each complete belief set either includes the fact $male(Floris)$ (with probability 0.4) or the fact $female(Floris)$ (with probability 0.6).

After adding the second formula, each complete belief set also contains one of $cs(Floris) \rightarrow male(Floris)$ and $cs(Floris) \rightarrow female(Floris)$. One of the resulting four belief sets, with probability $0.8 \cdot 0.6$, contains the formulas $female(Floris)$ and $cs(Floris) \rightarrow male(Floris)$.

Adding the third formula to the theory, the extension of the latter belief set allows one to infer $\neg male(Floris)$ and $\neg cs(Floris)$.

Belief sets in FOProbLog can be inconsistent. One should assign zero probability to such belief sets. With s a total choice and $c(s)$ ex-

¹ Katholieke Universiteit Leuven, email: firstname.lastname@cs.kuleuven.be

pressing consistency of s , $P(s|c(s))$, the *normalized* probability of a total choice s , is given by $P(s) \cdot P(c(s)|s) / P(c(s))$ where $P(c(s)|s)$ is either 1 or 0.

Ex. 2 Using all four formulas of Example 1 makes the belief set with formulas $\text{female}(\text{Floris})$, $\text{cs}(\text{Floris})$ and $\text{cs}(\text{Floris}) \rightarrow \text{male}(\text{Floris})$ (as well as another one) inconsistent. The probability of an inconsistent belief set is $0.8 \cdot 0.6 + 0.4 \cdot 0.2 = 0.56$. Normalizing the probabilities of the two consistent belief sets, we obtain $0.4 \cdot 0.8 / (1 - 0.56) = 0.73$ for the one containing $\text{male}(\text{Floris})$ and 0.27 for the one with $\text{female}(\text{Floris})$. Using all our independent formulas about maleness, we thus derive that *Floris* is more likely male than female, contrary to the belief about the name *Floris*.

While the above example can be modeled in ProbLog, the probabilities inferred there will have a different meaning, as ProbLog adopts a different view on consistency. In a ProbLog program containing clauses $\text{male}(\text{Floris})$. with probability 0.4 and $\text{male}(X) : \neg \text{cs}(X)$. with probability 0.8, those clauses together define the *male* predicate and, by closed world assumption, also $\neg \text{male}$, the equivalent of *female*. There are 4 different belief sets, one with both clauses, two with one clause and one without clauses, but none with an inconsistent belief set where *Floris* is both male and not male. The probability of $\text{male}(\text{Floris})$ is thus $0.8 + 0.4 - 0.8 \cdot 0.4 = 0.88$ instead of 0.73.

In the rest of the paper, we use a more basic form of FOProbLog which, similarly to ProbLog, distinguishes *probabilistic facts* from the *logical part* of a theory.

Def. 1 (FOProbLog theory) A FOProbLog theory $T = (PF, \Phi)$ consists of a probabilistic part PF and a logical part Φ . Its predicates are split into sets Σ_P and Σ_L of probabilistic and logical predicates, respectively. PF contains probabilistic facts of the form $\text{pf}(\bar{x}) : \alpha$, with α a non-zero probability, \bar{x} n different variables and $\text{pf}/n \in \Sigma_P$. Each ground instance $\text{pf}(\bar{x})\theta$ of such a fact is true with probability α , and is not a grounding of any other probabilistic fact. Different instances (of the same or of different probabilistic facts) are probabilistically independent. An assignment of a truth value to a ground instance of a probabilistic fact is called *atomic choice*, an assignment to all of them a *total choice*. The probability $\text{prob}(s)$ of a total choice s is the product of all α_i for true instances $\text{pf}_i(\bar{x})\theta : \alpha_i$ and all $(1 - \alpha_j)$ for false instances $\text{pf}_j(\bar{x})\theta : \alpha_j$. The logical part Φ of the theory consists of a set of implications $\forall \bar{x}. P(\bar{x}) \rightarrow F(\bar{x})$, where $F(\bar{x})$ is a first-order formula with free variables \bar{x} using only predicates from Σ_L , and $P(\bar{x})$ is a conjunction of literals with predicates from Σ_P^2 .

Ex. 3 Formulas (1)-(4) of Example 1 are now written as

- (1a) $\text{pf_mn}(\text{Floris}) : 0.4$
- (1b) $\text{pf_mn}(\text{Floris}) \rightarrow \text{male}(\text{Floris})$
- (1c) $\neg \text{pf_mn}(\text{Floris}) \rightarrow \text{female}(\text{Floris})$
- (2a) $\text{pf_cs}(x) : 0.8$
- (2b) $\forall x. \text{pf_cs}(x) \rightarrow (\text{cs}(x) \rightarrow \text{male}(x))$
- (2c) $\forall x. \neg \text{pf_cs}(x) \rightarrow (\text{cs}(x) \rightarrow \text{female}(x))$
- (3) $\forall x. \neg (\text{male}(x) \wedge \text{female}(x))$
- (4) $\text{cs}(\text{Floris})$

The probabilistic part of formulas (3) and (4) is true and omitted.

² A formula $\forall \bar{x}. \Psi_1 : \alpha_1 \vee \dots \vee \Psi_n : \alpha_n$ is split into n formulas $\forall \bar{x}. P_i(\bar{x}) \rightarrow \Psi_i(\bar{x})$ and $n - 1$ probabilistic facts are introduced, their probabilities are computed and the probabilistic part $P_i(\bar{x})$ of each formula contains a conjunction of probabilistic literals as described in [4].

To define the semantics of our logic, we fix a domain D and consider the set \mathcal{W}_D of all possible worlds in D , i.e., each I in \mathcal{W}_D is an interpretation for the vocabulary of the theory with D as its domain; in particular I assigns *true* or *false* to each probabilistic and each logical ground atom.

Notation 1 In what follows, we use $I \models s$ with I an interpretation and s a total choice to express that I makes the same truth assignments to the probabilistic facts as s . We often say I extends s .

Def. 2 (Semantics) Let $T = (PF, \Phi)$ be a theory and D a domain.

Let Cons be the set of total choices s such that there exists an interpretation I that extends s ($I \models s$) and is a model of Φ ($I \models \Phi$), i.e., the total belief set corresponding to the total choice is consistent.

$\widehat{\text{prob}}(s)$ (the normalized probability of a total choice) is given by $\text{prob}(s) \cdot \text{prob}(s \in \text{Cons}|s) / \sum_{s \in \text{Cons}} \text{prob}(s)$ where $\text{prob}(s \in \text{Cons}|s)$ is 1 when s is consistent and 0 otherwise.

A probability distribution μ over \mathcal{W}_D is a model of T , denoted $\mu \models T$, if and only if (i) for all $I : I \not\models \Phi$ implies $\mu(I) = 0$; (ii) for each total choice $s : \widehat{\text{prob}}(s) = \sum_{I \models s} \mu(I)$.

The theory is inconsistent when no consistent total choice exists. In that case, no probability distribution is defined.

Note that the normalization introduced here corresponds to conditioning on total choices with consistent extensions, which will be exploited in Section 3 to calculate probabilities of queries.

In contrast to the above informal exposition, where belief sets contained logical formulas and hence had interpretations over the logical predicates only, now interpretations assign truth to both probabilistic and logical predicates.

Ex. 4 In our example, with two probabilistic and three logical predicates and a single constant, we now have 32 possible interpretations. However, as before, only two of the four total choices can be extended in a belief set that is a model of the logical part; moreover, these models are unique. One of them is the belief set $\{\text{pf_cs}(\text{Floris}), \text{pf_mn}(\text{Floris}), \text{cs}(\text{Floris}), \text{male}(\text{Floris})\}$. It extends the total choice $\{\text{pf_cs}(\text{Floris}), \text{pf_mn}(\text{Floris})\}$, is a model of the logical part of the theory and is assigned probability 0.73 in the probability distribution that is a model. The other one, $\{\text{cs}(\text{Floris}), \text{female}(\text{Floris})\}$, extends the total choice where both probabilistic atoms are false; it is assigned probability 0.27. The total choices that make one probabilistic atom true and the other one false cannot be extended in a model of the logical part; their probability mass has been redistributed over the other ones.

In the above example, there is at most one belief set that extends a total choice and hence only one probability distribution that is a model. This does not hold in general. FOProbLog theories impose constraints, and any distribution satisfying these constraints is a model. Instead of making additional assumptions (such as the principle of indifference) to choose a specific distribution, we will restrict ourselves to sound inference, and will infer probability intervals only.

Ex. 5 Consider a theory consisting of the probabilistic fact $\text{pf}(x) : 0.7$ and the logical formula $\forall x. \text{pf}(x) \rightarrow p(x)$ with a single element domain $\{A\}$. The total choice $\text{pf}(A)$ can be extended into the belief set $\{\text{pf}(A), p(A)\}$ and hence $\mu(\{\text{pf}(A), p(A)\}) = 0.7$. The empty total choice can be extended in two ways, namely \emptyset and $\{p(A)\}$; hence $\mu(\emptyset) + \mu(\{p(A)\}) = 0.3$. Any distribution that satisfies the latter constraint is a model. The principle of indifference would assign probability 0.15 to each of the latter two belief sets and hence 0.85 to $p(A)$ and 0.15 to $\neg p(A)$. We infer that the probability of $p(A)$ is in the interval $[0.7, 1.0]$ and that of $\neg p(A)$ in $[0.0, 0.3]$.

3 Inference

We are now interested in deciding what the theory T allows us to conclude over the probability of some query formula Q . For each given domain, T has a non-empty set $\hat{\mathcal{M}}$ of models μ . Each $\mu \in \hat{\mathcal{M}}$ assigns a particular probability $\mu(Q) = \sum_{I \models Q} \mu(I)$ to Q , yielding, in general, a non-empty probability interval $[\min_{\mu \in \hat{\mathcal{M}}} \mu(Q), \max_{\mu \in \hat{\mathcal{M}}} \mu(Q)]$. We are now interested in the inference task of determining this interval. Because $\max_{\mu \in \hat{\mathcal{M}}} \mu(Q)$ must be equal to $1 - \min_{\mu \in \hat{\mathcal{M}}} \mu(\neg Q)$, we can restrict attention to the computation of a lower bound on the probability of a query. As the following theorem shows, we can compute this lower bound without having to consider any specific model μ of T .

Theorem 1 *Let $\hat{\mathcal{M}}$ be the non-empty set of models of a consistent theory T . Then $\min_{\mu \in \hat{\mathcal{M}}} \mu(Q) = \sum_{s \models Q} \widehat{prob}(s)$, where $s \models Q$ means that Q holds in all $I \in \mathcal{W}_D$ such that $I \models s$.*

Proof 1 *Let $W_Q = \{I \mid I \models Q\}$ and $W_{SQ} = \{I \mid \exists s : I \models s \wedge s \models Q\}$. By definition, $\mu(Q) = \sum_{I \in W_Q} \mu(I)$. Also by definition, $\sum_{s \models Q} \widehat{prob}(s) = \sum_{I \in W_{SQ}} \mu(I)$. Obviously, $\sum_{I \in W_Q} \mu(I) \geq \sum_{I \in W_{SQ}} \mu(I)$. To prove the theorem, it suffices to show that there exists a probability distribution μ' that is a model and for which the equality holds. The equality holds if μ' assigns 0 probability to all $I \in W_Q \setminus W_{SQ}$. For such I , we can distinguish two cases. Either I restricted to the probabilistic atoms gives a total choice that cannot be extended into a consistent belief set, in which case all distributions that are a model assign 0 probability to I . Or I restricted to the probabilistic atoms gives a total choice s with at least one I' extending s such that $I' \not\models Q$. In this case, μ' can be chosen to assign all the probability mass $\widehat{prob}(s)$ to interpretations that are not models of Q and hence set $\mu'(I) = 0$. Hence $\mu'(I) = 0$ for all $I \in W_Q \setminus W_{SQ}$.*

The normalized probability of a total choice that can be extended into a consistent belief set is obtained by dividing its probability by the probability of all such total choices. The latter is the complement of the probability of those total choices where this is not possible, that is, the belief sets containing *false*. Furthermore, recall that $\widehat{prob}(s) = 0$ for total choices without consistent extension. Hence, we also have $\min_{\mu \in \hat{\mathcal{M}}} \mu(Q) = \left(\sum_{s \models Q \wedge \neg \text{false}} \text{prob}(s) \right) / \left(1 - \sum_{s \models \text{false}} \text{prob}(s) \right)$, which in fact corresponds to the conditional probability $\text{prob}(Q \mid \neg \text{false})$.

Ex. 6 *Let us reconsider the theory consisting of $pf(x) : 0.7$ and $\forall x. pf(x) \rightarrow p(x)$. The query $p(A)$ can be proven with a total choice that includes the probabilistic fact $pf(A)$. As no total choice results in inconsistency, this gives a minimal probability of 0.7. No proofs are possible for $\neg p(A)$, hence the maximal probability of $p(A)$ is 1.*

The number of total choices is exponential in the number of probabilistic facts, hence it is not feasible to evaluate a query for each total choice. More promising is the ProbLog approach that enumerates all proofs in terms of the probabilistic facts they use and encodes this information in a Binary Decision Diagram (BDD), which can be used to compute the probability of the query [5]. The main difference is that we do not work with Horn clauses but with full first order logic, hence we cannot, as in ProbLog, use the SLD proof procedure to enumerate proofs. However, for quickly building a first prototype and for making maximal use of the ProbLog technology, it is interesting to stay as much as possible within Prolog. Here the work of Stickel [15]

that describes how to use Prolog technology for building a first order logic theorem prover comes to the rescue and allows us to convert FOProbLog theories into ProbLog programs.

The basic idea of Stickel's work is to transform formulas into clausal form and to encode each n -literal clause $l_1 \vee \dots \vee l_n$ by n Horn clauses of the form $l_i : \neg \text{not}(l_1), \dots, \text{not}(l_{i-1}), \text{not}(l_{i+1}), \dots, \text{not}(l_n)$. To obtain Horn clauses, negative literals are encoded by positive ones: For each predicate p/n , Stickel introduces a not_p/n and the negation of $p(\bar{t})$ is replaced by $\text{not}_p(\bar{t})$. In the context of FOProbLog, for each formula $\forall \bar{x}. P(\bar{x}) \rightarrow F(\bar{x})$, the logical part $F(\bar{x})$ is converted in clausal form, Stickel's transformation is applied to the resulting clauses, and the conjunction $P(\bar{x})$ is added to the bodies of the final Horn clauses.

Ex. 7 *Applying the transformation to our example gives the following ProbLog program (we switch to Prolog notation for constants and variables):*

```
(1a) 0.4::pf_mn(floris)
(1b) male(floris) :- pf_mn(floris).
(1c) female(floris) :- not(pf_mn(floris)).
(2a) 0.8::pf_cs(X)
(2b) male(X) :- cs(X), pf_cs(X).
(2b) not_cs(X) :- not_male(X), pf_cs(X).
(2c) female(X) :- cs(X), not(pf_cs(X)).
(2c) not_cs(X) :- not_female(X), not(pf_cs(X)).
(3a) not_female(X) :- male(X).
(3b) not_male(X) :- female(X).
(4) cs(floris).
```

The transformation also has to generate clauses with *false* in the head, as those are needed during inference to take consistency into account. One way to do so is to use the knowledge that a resolution proof must use at least one clause $pf(\bar{x}) \rightarrow \text{not}(l_1) \vee \dots \vee \text{not}(l_n)$ with only negative literals (to have all such clauses as "set of support"). By adding *false* : $\neg l_1, \dots, l_n, pf(\bar{x})$ for each such clause, we ensure that all proofs of *false* employ such clauses. Alternatively, one can use the positive clauses (clauses with only positive literals) as set of support. Simply adding a clause *false* : $\neg p(\bar{X}), \text{not}_p(\bar{X})$ for each logical predicate p/n is also possible, but will typically result in a lot of redundant proofs.

Ex. 8 *In our example, using negative clauses as set of support would add:*

```
false :- male(X), female(X).
whereas using positive clauses would add:
```

```
false :- not_cs(floris).
false :- not_male(floris), pf_mn(floris).
false :- not_female(floris), not(pf_mn(floris)).
```

Stickel additionally needs to modify certain parts of the inference mechanism. First, Prolog's input resolution is extended with ancestor resolution: resolution between the current goal and one of the ancestors in the linear chain from query to current goal. A convenient way to do so in the setting of Prolog's depth first left to right execution policy is to keep track of the selected literals with uncompleted proof. If in the uncompleted proof of a literal $p(\bar{t})$ (or $\text{not}_p(\bar{t})$), its negation $\text{not}_p(\bar{s})$ (or $p(\bar{s})$) is selected, the latter literal can be resolved by unifying its atom with the atom of the former (i.e. unifying \bar{t} with \bar{s}) [15]. Also, care is required to avoid unsound unification. In general, some unifications may have to be replaced by a sound variant that performs an occur check. When proving queries in knowledge bases, it is unlikely that the occur check is needed. Finally, performing iterative deepening avoids that the search gets trapped in an infinite branch.

```

¬sameBib(b1, b2)
¬sameAuthor(a1, a2)
∀ b1, b2, b3 : sameBib(b1, b2) ∧ sameBib(b2, b3) →
  sameBib(b1, b3)
∀ b1, b2, a1, a2 : author(b1, a1) ∧ author(b2, a2) ∧
  sameAuthor(a1, a2) → sameBib(b1, b2)
∀ a1, a2, w : hasWordAuthor(a1, w) ∧
  hasWordAuthor(a2, w) → sameAuthor(a1, a2)
∀ a1, a2, w : ¬hasWordAuthor(a1, w) ∧
  hasWordAuthor(a2, w) → ¬sameAuthor(a1, a2)

```

Figure 1. A key part of the logical theory.

Given the transformed theory and a query Q , we can now use ProbLog’s machinery to construct two Boolean formulas ψ and ϕ representing the proofs of Q and the proofs of *false*, respectively. The formula $\psi \wedge \neg\phi$ thus restricts the proofs of Q to consistent belief sets, whereas ϕ is used for normalization, i.e. $\min_{\mu \in \mathcal{M}} \mu(Q)$ is obtained by $P(\psi \wedge \neg\phi) / (1 - P(\phi))$, where both intermediate results are calculated using ProbLog’s BDD algorithm. For the running example, we obtain $\phi = (\neg pf_cs(floris) \wedge pf_mn(floris)) \vee (pf_cs(floris) \wedge \neg pf_mn(floris))$, and, for the query $? - male(floris)$, $\psi \wedge \neg\phi = pf_cs(floris) \wedge pf_mn(floris)$.

4 A case study

To assess performance in practice, we explored an application of Markov Logic in entity resolution [14]. We will first review some details of the application, and then describe how to encode key parts of it directly in ProbLog.

In this application, a fact database containing information about bibliographic entries such as authors and titles is combined with a probabilistic first order theory describing when two database references of the same type are likely the same. The purpose of entity resolution is to compute the probability that two keys or two authors (e.g. `author_william_w_cohen_` and `author_w_w_cohen_`) refer to the same object, i.e., to be able to get probabilities for the queries *sameBib*($b1, b2$) and *sameAuthor*($a1, a2$), as well as for *not_sameBib*($b1, b2$) and *not_sameAuthor*($a1, a2$).

Part of the facts database is a binary encoding of a ternary relation between authors, paper title and publication venue. A tuple (*author, paper, venue*) is encoded as *author*(*bib, author*), *title*(*bib, paper*) and *venue*(*bib, venue*) with *bib* an arbitrary key identifying a bibliographic entry. As formulas for authors, papers and venues are similar in structure, we will restrict our discussion to those concerning authors and bibliographic entries. Authors are strings (extracted from web pages) that are composed from different words (e.g. `author_william_w_cohen_` and `author_w_w_cohen_` are composed of the words `word_cohen`, `word_w`, and `word_william`). This information is encoded in the database relation *hasWordAuthor*(*author, word*).

Figure 1 shows the first order logic formulas used in [14] to model a key part of the application. A first simple set of rules states that different strings refer to different authors or different bibliographical

```

not_author(B, A) :- not(author(B, A)).
not_hasWordAuthor(A, W) :-
  not(hasWordAuthor(A, W)).

false :- author(B, A), not_author(B, A).
false :- hasWordAuthor(A, W),
  not_hasWordAuthor(A, W).

sameBib(B, B).
sameAuthor(A, A).
not_sameBib(B1, B2) :- B1 \= B2, pf1(B1, B2).
not_sameAuthor(A1, A2) :- A1 \= A2, pf2(A1, A2).

false :- domBib(B), not_sameBib(B, B).
false :- domAuthor(A), not_sameAuthor(A, A).

sameBib(B1, B3) :- sameBib(B1, B2),
  sameBib(B2, B3), pf5(B1, B2, B3).
not_sameBib(B1, B2) :- sameBib(B2, B3),
  not_sameBib(B1, B3), pf5(B1, B2, B3).
not_sameBib(B2, B3) :- sameBib(B1, B2),
  not_sameBib(B1, B3), pf5(B1, B2, B3).

sameBib(B1, B2) :-
  author(B1, A1), author(B2, A2),
  sameAuthor(A1, A2), pf9(B1, B2, A1, A2).
not_sameAuthor(A1, A2) :-
  author(B1, A1), author(B2, A2),
  not_sameBib(B1, B2), pf9(B1, B2, A1, A2).
not_author(B1, A1) :-
  sameAuthor(A1, A2), author(B2, A2),
  not_sameBib(B1, B2), pf9(B1, B2, A1, A2).
not_author(B2, A2) :-
  sameAuthor(A1, A2), author(B1, A1),
  not_sameBib(B1, B2), pf9(B1, B2, A1, A2).

```

Figure 2. Key clauses in ProbLog.

entries, respectively. The next formula is an example of rules expressing that the relations about sameness are likely transitive, while the remaining ones link authors to bibliographical entries and words to authors, respectively.

Figure 2 shows the result of translating those formulas into ProbLog. We cannot blindly translate the first order theory. The main reason is that the closed world assumption is applied on the large database. Explicitly adding all negative facts would result in an exponential blow-up in the size of the logical theory. Therefore, we instead use negation as failure to encode them and add the first group of rules to our theory. Note that calls to these predicates are correctly executed only when all arguments are ground. This need not be a problem because ProbLog aims at inference of ground queries. Another difficulty – if we stick to the negative clauses as set of support – is that, for each tuple (b, a) not in the database, we would need to add a clause *false* :- *author*(b, a). As we cannot enumerate all such pairs (b, a), we resort to a different approach to define *false*. Using a positive set of support, we should add *false* :- *not_author*(b, a) for each pair in the author relation. As the theory does not contain clauses with *author*/2 in the head, we can use the author relation to generate the pairs. Furthermore, this approach introduces less redundancy in the search space of proofs for *false* than the approach of simply adding a rule *false* :- $p(\bar{X}), not_p(\bar{X})$ for each of the predicates *author*/2, *hasWordAuthor*/2, *sameAuthor*/2, and *sameBib*/2. The next block of rules states that identical strings refer to identical entities (with certainty), different strings to different entities (with a rather high probability). As we have added positive clauses, we also have to add the next block of extra rules for proving

false. Here `domBib/1` and `domAuthor/1` are used to generate the values for the keys and the authors respectively (appropriate ground facts have to be added to the database). Each clause about transitivity gives rise to three ProbLog clauses, which form the next block. Note that loop checking as well as ancestor resolution are important for doing exact inference with these clauses. The clauses linking authors to bibliographic entries give rise to the last block. Finally, as the rules linking authors and words can be translated in the same way as the previous clauses, we do not further elaborate them.

However, one important observation is that the sharing of words in different author names, paper titles, and venues, results in a densely connected network to the point that almost for every pair (b_1, b_2) of keys, one can prove *sameBib* (b_1, b_2) as well as *not_sameBib* (b_1, b_2) . Hence, there are a lot of inconsistent total choices, it is essential to perform normalization and to execute the *false* query. Clearly, this is very demanding.

5 Experiments

We set up experiments to investigate the feasibility of inference in FOProbLog. We focus on the query *false*, as this can involve all possible queries in a theory and thus is the most challenging query. We used a version of ProbLog with tabling [6] which we extended with ancestor resolution and iterative deepening as suggested by Stickel [15]. Experiments are performed on an Intel Core 2 Duo CPU at 3.00GHz with 2GB RAM running Ubuntu 8.04.2 Linux. In our experiments ProbLog is used to first collect the proofs and construct a Boolean formula for *false* and to then compute the probability by constructing a BDD.

The first experiment uses the following FOProbLog theory

$$\forall x, y, z. pf1(x, y, z) \rightarrow (Fr(x, y) \wedge Fr(y, z) \rightarrow Fr(x, z))$$

$$\forall x. pf2(x) \rightarrow (Sm(x) \rightarrow Ca(x))$$

$$\forall x, y. pf3(x, y) \rightarrow (Fr(x, y) \rightarrow (Sm(x) \leftrightarrow Sm(y)))$$

and randomly generated databases to investigate the influence of the domain size and the maximum depth used in iterative deepening. The latter limits transitive closure. The runtimes to obtain the Boolean formula representing *false* are presented in Figure 3; a missing bar indicates that the experiment failed to properly terminate. We notice that the search space grows exponentially with the depth limit, while the influence of the domain size is less drastic. Assessing the probability of the Boolean formula through BDDs is feasible within one minute for most of these experiments.

The second experiment uses the entity resolution model of Section 4 with the full database of [14] containing 1295 bibliographic entries involving roughly 90 authors, 400 venues, 200 titles and 2700 words. Figure 4 shows the time to obtain the Boolean formula representing *false* for increasing maximum search depth. The results confirm that search time increases exponentially, as can be expected for such a densely connected problem. The limiting factor of our current prototype is the size of the resulting BDDs; in this application, they are too large to be constructed within a time limit of one hour.

It is worth mentioning that in both experiments, the Boolean formulas for other queries are far smaller. However, in most cases, the time still increases exponentially with the depth bound. Furthermore, our current results do not permit general conclusions about the gap between minimal and maximal probabilities obtained for each query.

6 Discussion

In FOProbLog, probabilities are associated with formulas in first order logic. The assumption that these formulas are independent allows

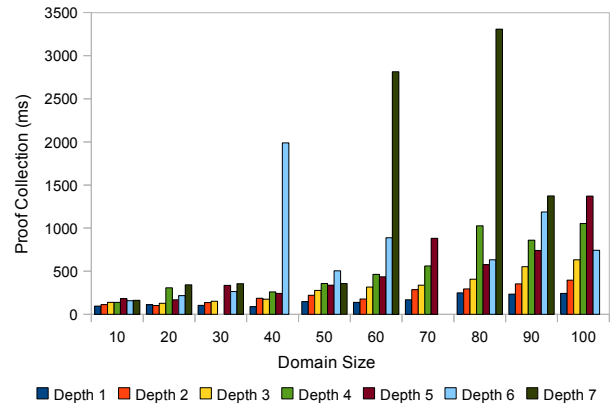


Figure 3. Friends experiments.

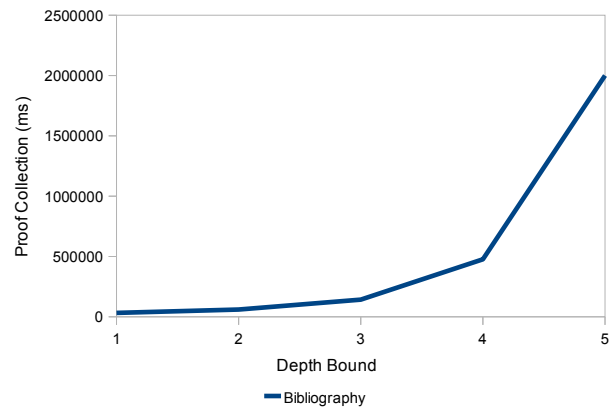


Figure 4. Bibliography experiments.

us to define a semantics based on complete belief sets. Inference in FOProbLog is the task of calculating the probability that a query can be proven in a randomly selected complete belief set. Randomly selected belief sets can be inconsistent. The probability of such a selection can be computed. Normalization assigns 0 probability to inconsistent total choices and redistributes their probability mass over the consistent ones. The probability of the truth of a query is computed with respect to consistent total choices.

One could argue that inconsistency is an indication of the violation of the independence assumption. Interestingly, inconsistent belief sets typically arise after adding factual knowledge (adding *cs(Floris)* or adding the database of bibliographical information). This factual knowledge can be seen as *evidence* that excludes certain belief sets. Inconsistency in the non-factual part is thus not an indication of poor design, as it can make sense to add a formula as evidence to an existing theory. This refines the theory by ruling out certain complete belief sets, and hence causes a redistribution of the probability mass.

A related issue is the presence of redundant formulas. One might consider a situation where two independent experts contribute exactly the same formula to a theory, each of them claiming this formula with probability p_1 and p_2 respectively. The probability that

this formula then holds in a complete belief set is $p_1 + p_2 - p_1 \cdot p_2$. This makes sense when the expertise is really based on different knowledge. However, with each additional independent expert coming up with the same formula and assigning some probability p_i to it, the overall probability would further increase (unless the experts explicitly assign a probability to both the formula and its negation). It is plausible that different experts have used the same common knowledge to come up with their formula and hence the independence assumption is violated. In general, the interaction can be more subtle. For example, in the context of our case study, one could add that *sameBib* is a symmetrical relation. This will likely result in different probabilities for queries³.

One should be aware that the probability annotating a formula is the probability that a ground instance is included in a belief set. To know the minimal probability that the formula can be proven in a randomly selected consistent belief set, one should query for it. This probability can be different for different instances and our current prototype supports querying only for ground instances.

In reality, given a theory and a number of datasets with evidence, one will typically learn probabilities. In that case it is very desirable to have a logical part that avoids inconsistencies and redundancies as much as possible. Indeed, consider again the extreme case that one has two copies of the same formula in the theory. From the evidence, one will derive only the value of $p_1 + p_2 - p_1 \cdot p_2$, so there is a certain randomness in allocating p_1 and p_2 . Hence, the logical theory better avoids inconsistencies and redundancies to facilitate the understanding of the learned probabilities.

7 Related Work and Conclusion

We have proposed FOProbLog, a simple but very expressive probabilistic logic and defined its semantics. As Markov Logic, FOProbLog is based on a notion of soft constraint, but formulas are labeled by probabilistic predicates instead of weights. Indeed, the higher the probability of a formula, the lower the probability will be of interpretations not satisfying it. The use of probabilities should make FOProbLog more intuitive from a knowledge representation point of view. Another difference is that the semantics of Markov Logic is defined through the sets of weighted ground instances of the formulas, while FOProbLog's semantics is defined in terms of groundings of probabilistic facts. The difference can be illustrated using the formula $\forall X.pf(X) \rightarrow \exists Y.likes(X, Y)$. In Markov Logic the probability would necessarily depend on the groundings of both X and Y , whereas in FOProbLog it only depends on those of X . Furthermore, the key inference mode in Markov Logic is typically based on the MPE principle approximating a most likely state given some evidence, while in FOProbLog an interval on the probability of queries is computed. In this regard, FOProbLog is also related to the work on Probabilistic Logic Programming (PLP) [8]. Both PLP and FOProbLog combine probabilities of basic random events to assign probabilities to Boolean valued interpretations, subject to constraints given by the program or theory. However, in PLP, basic random events as well as individual atoms in clauses can be annotated with (independent) closed probability intervals. Inference in PLP obtains maximally concise probability bounds for a given query by propagating those intervals using techniques from linear programming. Finally, FOProbLog is also closely related to systems such as PRISM [13] and ICL [11], which are both based on atomic and total choices, but only for definite clauses, as well as to Stochastic Logic

³ We have followed as close as possible the original Markov Logic formulation as we only wanted to evaluate the feasibility of inference.

Programs [7], an extension of probabilistic context-free grammars, where normalization is required to redistribute the probability mass of failing derivations over successful ones.

We have described how FOProbLog theories can be transformed into ProbLog programs and have identified the bottleneck to do inference in problems similar to the ones tackled by Markov Logic [12]. Much work remains to be done. Tabling makes search for all proofs feasible even for large problems. But calculating the BDD needed for normalization becomes too expensive. Hence approximation methods for tabled ProbLog are a first promising direction. Another one is the analysis of independencies in the theory, which may allow to restrict consistency computations to the parts of the database influencing the probability of the current query. Similarly, studying the influence of the depth bound on the probability values obtained may provide insights into the necessary depth of search. So far we only compute probabilities of ground atomic queries. The question arises whether inference for other queries is feasible. Our logic is as expressive as Nilson's [9], indeed, for a formula F we can write $F : \alpha \vee \neg F : 1 - \alpha$, so it is natural that some queries are hard to evaluate. This has to be analysed. On the representation side, the insights obtained from the case study can serve as basis for an automated translation of FOProbLog into ProbLog. Finally, learning parameters for FOProbLog based on corresponding techniques for ProbLog [2] is another interesting line of work.

Acknowledgments. This work is supported by GOA/08/008 Probabilistic Logic Learning. AK and BG are supported by FWO-Vlaanderen. JV is a postdoctoral researcher of FWO-Vlaanderen.

REFERENCES

- [1] *An Introduction to Statistical Relational Learning*, eds., L. Getoor and B. Taskar, MIT Press, 2007.
- [2] B. Gutmann, A. Kimmig, L. De Raedt, and K. Kersting, 'Parameter learning in probabilistic databases: A least squares approach', in *ECML PKDD 2008*, pp. 473–488. Springer, (2008).
- [3] K. Kersting and L. De Raedt, 'Basic principles of learning Bayesian logic programs', in *Probabilistic Inductive Logic Programming*, ed., L. De Raedt *et al.*, 189–221, Springer, (2008).
- [4] A. Kimmig, B. Gutmann, and V. Santos Costa, 'Trading memory for answers: Towards tabling ProbLog', in *Int. Workshop on Statistical Relational Learning*, eds., P. Domingos and K. Kersting, (2009).
- [5] A. Kimmig, V. Santos Costa, R. Rocha, B. Demoen, and L. De Raedt, 'On the efficient execution of ProbLog programs', in *ICLP*, pp. 175–189, (2008).
- [6] T. Mantadelis and G. Janssens, 'Tabling relevant parts of SLD proofs for ground goals in a probabilistic setting', in *CICLOPS*, eds., P. Tarau, P. Moura, and N. Zhou, (2009).
- [7] S. H. Muggleton, 'Stochastic logic programs', in *Advances in Inductive Logic Programming*, ed., L. De Raedt, IOS Press, (1996).
- [8] R. T. Ng and V. S. Subrahmanian, 'Probabilistic logic programming', *Inf. Comput.*, **101**(2), 150–201, (1992).
- [9] N. J. Nilson, 'Probabilistic logic', *AI*, **28**, 71–87, (1986).
- [10] D. Poole, 'Abducting through negation as failure: stable models within the independent choice logic', *Journal of Logic Programming*, **44**, 5–35, (2000).
- [11] D. Poole, 'The independent choice logic and beyond', in *Probabilistic Inductive Logic Programming*, ed., L. De Raedt *et al.*, 222–243, Springer, (2008).
- [12] M. Richardson and P. Domingos, 'Markov logic networks', *Machine Learning*, **62**(1-2), 107–136, (2006).
- [13] T. Sato and Y. Kameya, 'PRISM: A language for symbolic-statistical modeling', in *IJCAI*, pp. 1330–1339, (1997).
- [14] P. Singla and P. Domingos, 'Entity resolution with markov logic', in *ICDM*, pp. 572–582, (2006).
- [15] Mark E. Stickel, 'A Prolog technology theorem prover: implementation by an extended Prolog compiler', *Journal of Automated Reasoning*, **4**, 353–380, (1988).