A Distribution Semantics for Probabilistic Ontologies

Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi, and Simone Albani

ENDIF – University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy
{elena.bellodi,evelina.lamma,fabrizio.riguzzi}@unife.it
simone.albani@student.unife.it

Abstract. We present DISPONTE, a semantics for probabilistic ontologies that is based on the distribution semantics for probabilistic logic programs. In DISPONTE each axiom of a probabilistic ontology is annotated with a probability. The probabilistic theory defines thus a distribution over normal theories (called worlds) obtained by including an axiom in a world with a probability given by the annotation. The probability of a query is computed from this distribution with marginalization. We also present the system BUNDLE for reasoning over probabilistic OWL DL ontologies according to the DISPONTE semantics. BUNDLE is based on Pellet and uses its capability of returning explanations for a query. The explanations are encoded in a Binary Decision Diagram from which the probability of the query is computed.

1 Introduction

Representing probabilistic knowledge and reasoning with it is fundamental in order to realize the full vision of the Semantic Web, due to the ubiquity of uncertainty in the real world and on the Web [24]. Various authors have advocated the use of probabilistic ontologies, see e.g. [17], and many proposals have been put forward for allowing ontology languages, and OWL in particular, to represent uncertainty.

Similarly, in the field of logic programming, there has been much work on introducing uncertainty in the programs. Among the various proposals, the distribution semantics [22] has emerged as one of the most effective approaches and it underlies many languages such as PRISM [22], ICL [19], Logic Programs with Annotated Disjunctions [26] and ProbLog [3]. In this semantics a probabilistic logic program defines a probability distribution over a set of normal logic programs (called *worlds*). The distribution is extended to a joint distribution over worlds and queries; the probability of a query is obtained from this distribution by marginalization. In general, the problem of integrating logic and probability has been much studied lately, with proposals such as Markov Logic [20], Multi Entity Bayesian Networks [12] and Probabilistic Relational Models [10].

In this paper we propose to apply this approach to ontology languages and, in particular, to the OWL DL fragment, that is based on the description logic $\mathcal{SHOIN}(\mathbf{D})$. However, the approach is applicable in principle to any description

logic. We called the approach DISPONTE for "DIstribution Semantics for Probabilistic ONTologiEs" (Spanish for "get ready"). The idea is to annotate each axiom of a theory with a probability and assume that each axiom is independent of the others. A probabilistic theory defines thus a distribution over normal theories (worlds) obtained by including an axiom in a world with a probability given by the annotation. The probability of a query is again computed from this distribution with marginalization.

We also present the system BUNDLE for "Binary decision diagrams for Uncertain reasoNing on Description Logic thEories" that performs inference over probabilistic OWL DL ontologies. BUNDLE uses the inference techniques developed for probabilistic logic programs under the distribution semantics [8,21] and, in particular, the use of Binary Decision Diagrams (BDDs) for encoding explanations to queries and for computing their probability.

BUNDLE is based on the Pellet reasoner [23] for OWL DL and exploits its capability of returning explanations for queries in the form of a set of sets of axioms from which BUNDLE builds a BDD for computing the probability. In this way we provide an effective reasoning system for DISPONTE.

The paper is organized as follows. Section 2 describes the distribution semantics for logic programs while Section 3 presents DISPONTE. Section 4 illustrates BUNDLE and Section 5 discusses current limitations of DISPONTE and BUNDLE. Section 6 describes related works while Section 7 concludes the paper.

2 The Distribution Semantics in Probabilistic Logic Programming

The probabilistic logic programming languages based on the distribution semantics differ in the way they define the distribution over logic programs. Each language allows probabilistic choices among atoms in clauses. Let us consider ProbLog [3] which is the language with the simplest syntax. A ProbLog program T is composed of a normal logic program T_C and a set of probabilistic facts T_P. Each probabilistic fact is of the form $p_i :: F_i$. where p_i is a probability (i.e. $p_i \in [0, 1]$) and F_i is a atom. This means that every grounding of F_i is a Boolean random variable that assumes *true* value with probability p_i and *false* with probability $1 - p_i$.

Let us call T_F the set of atoms obtained by removing the probabilistic annotation from the probabilistic facts. Let us consider the case in which $T_C \cup T_F$ does not contain function symbols so that its Herbrand base is finite. Let us call $ground(T)$ the grounding of a normal program T. Since there are no function symbols, $ground(T_C \cup T_F)$ is finite and so is the grounding $ground(T_F)$ obtained by grounding the probabilistic atoms with constants from the Herbrand universe of $T_C \cup T_F$. So each probabilistic fact F_i has a finite set of groundings.

A substitution is a set of couples V/c where V is a variable and c is a constant. A substitution θ_j is applied to a logic atom F, indicated with $F\theta_j$, by replacing the variables in the substitution with constants. A substitution θ_j is grounding for logic atom F if $F\theta_j$ is ground. Suppose that a grounding is obtained with

the substitution θ_j: $F_i\theta_j$ corresponds to a Boolean random variable X_{ij} that is independent of the others.

Example 1. The following ProbLog program T encodes a very simple model of the development of an epidemic or pandemic:

$C_1 = epidemic : -flu(X), epid(X), cold.$
$C_2 = pandemic : -flu(X), \backslash + epid(X), pand(X), cold.$
$C_3 = flu(david).$
$C_4 = flu(robert).$
$F_1 = 0.7 :: cold.$
$F_2 = 0.6 :: epid(X).$
$F_3 = 0.3 :: pand(X).$

This program models the fact that if somebody has the flu and the climate is cold there is the possibility that an epidemic or a pandemic arises. We are uncertain whether the climate is cold but we know for sure that David and Robert have the flu. $epid(X)$ and $pand(X)$ can be considered as "probabilistic activators" of the effects in the head given that the causes ($flu(X)$ and $cold$) are present. $\backslash + epid(X)$ means the negation of $epid(X)$.

Fact F_1 has only one grounding so there is a single Boolean variable X_{11}. Fact F_2 has two groundings, $epid(david)$ and $epid(robert)$ so there are two Boolean random variables X_{21} and X_{22}. F_3 also has two groundings so there are two Boolean random variables X_{31} and X_{32}.

In order to present the distribution semantics, let us first give some definitions. An *atomic choice* is a selection of a value for a grounding of a probabilistic fact F and is represented by the triple (F_i, θ_j, k) where θ_j is a substitution grounding F_i and $k \in \{0, 1\}$. A set of atomic choices κ is *consistent* if $(F_i, \theta_j, k) \in \kappa, (F_i, \theta_j, m) \in \kappa \Rightarrow k = m$, i.e., only one truth value is selected for a ground fact. A *composite choice* κ is a consistent set of atomic choices. The probability of composite choice κ is $P(\kappa) = \prod_{(F_i,\theta_j,1)\in\kappa} p_i \prod_{(F_i,\theta_j,0)\in\kappa}(1 - p_i)$. A *selection* σ is a total composite choice (one atomic choice for every grounding of every probabilistic fact). A selection σ identifies a normal logic program w_σ called a *world* in this way: $w_\sigma = T_C \cup \{F_i\theta_j | (F_i, \theta_j, 1) \in \sigma\}$. The probability of w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(F_i,\theta_j,1)\in\kappa} p_i \prod_{(F_i,\theta_j,0)\in\kappa}(1 - p_i)$. Since $ground(T_F)$ is finite the set of worlds is finite: $W_T = \{w_1, \ldots, w_m\}$ and $P(w)$ is a distribution over worlds: $\sum_{w\in W_T} P(w) = 1$. A world w_σ is *compatible* with a composite choice κ if $\kappa \subseteq \sigma$

We can define the conditional probability of a query Q given a world as $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. This allows to define a joint distribution of the query and the worlds $P(Q, w)$ by using the product rule of the theory of probability: $P(Q, W) = P(Q|w)P(w)$. The probability of Q can then be obtained from the joint distribution by the sum rule (marginalization over Q):

$$P(Q) = \sum_{w\in W_T} P(Q, w) = \sum_{w\in W_T} P(Q|w)P(w) = \sum_{w\in W_T : w\models Q} P(w) \qquad (1)$$

In Example 1, T has 5 Boolean random variables and thus 32 worlds. The query *epidemic* is true in 5 of them and its probability is $P(epidemic) = 0.588$.

It is often unfeasible to find all the worlds where the query is true so inference algorithms find instead *explanations* for the query [8,21] , i.e. composite choices such that the query is true in all the worlds that are compatible with them . For example, $\kappa_1 = \{(F_2, \{X/david\}, 1), (F_1, \{\}, 1)\}$ is an explanation for the query *epidemic* and so is $\kappa_2 = \{(F_2, \{X/robert\}, 1), (F_1, \{\}, 1)\}$.

Each explanation κ identifies a set of worlds, those that are compatible with it, and a set of explanations K identifies the set ω_K of worlds compatible with one of its explanations ($\omega_K = \{w_\sigma | \kappa \in K, \kappa \subseteq \sigma\}$). A set of explanations K is *covering* for a query Q if every world in which Q is true is in ω_K. For example, $K = \{\kappa_1, \kappa_2\}$ is covering for the query *epidemic*.

The probability of a query can thus be computed from a covering set of explanations for the query by computing the probability of the Boolean formula

$$B(Q) = \bigvee_{\kappa \in K} \bigwedge_{(F_i, \theta_j, 1) \in \kappa} X_{ij} \bigwedge_{(F_i, \theta_j, 0) \in \kappa} \neg X_{ij} \tag{2}$$

For Example 1, the formula is $B(epidemic) = X_{11} \wedge X_{21} \vee X_{11} \wedge X_{22}$.

Explanations however, differently from possible worlds, are not necessarily mutually exclusive with respect to each other, so the probability of the query can not be computed by a summation as in (1). In fact computing the probability of a DNF formula of independent Boolean random variables is a #P-complete problem [25]. The method that was found to be the most efficient up to now consists in building a Binary Decision Diagram for the formula and using a dynamic programming algorithm on the BDD [8,21]. A BDD is a rooted graph that has one level for each variable. Each node n has two children, a 0-child and a 1-child. The leaves store either 0 or 1. Given values for all the variables, a BDD can be used to compute the value of the formula by traversing the graph starting from the root, following the edges corresponding to the variables values and returning the value associated to the leaf that is reached. The BDD for Example 1 is shown in Figure 1.

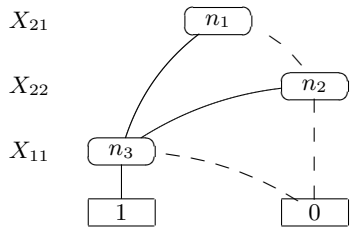

Fig. 1. BDD for Example 1.

A BDD performs a Shannon expansion of the Boolean formula $f(\mathbf{X})$, so that if X is the variable associated to the root level of a BDD, the formula $f(\mathbf{X})$ can be represented as $f(\mathbf{X}) = X \wedge f^X(\mathbf{X}) \vee \neg X \wedge f^{\neg X}(\mathbf{X})$ where $f^X(\mathbf{X})$

$(f^{\neg X}(\mathbf{X}))$ is the formula obtained by $f(\mathbf{X})$ by setting X to 1 (0). Now the two disjuncts are mutually exclusive and the probability of $f(\mathbf{X})$ can be computed as $P(f(\mathbf{X})) = P(X)P(f^X(\mathbf{X}))+(1-P(X))P(f^{\neg X}(\mathbf{X}))$ Figure 2 shows the function Prob that implements the dynamic programming algorithm for computing the probability of a formula encoded as a BDD.

Fig. 2. Function Prob: computation of the probability of a Boolean formula encoded as a BDD with root *node*.

```
1: function Prob(node)
2:     if node is a terminal then
3:         return value(node)                          ▷ value(node) is either 0 or 1
4:     else
5:         let X be v(node)                ▷ v(node) is the variable associated to node
6:         return Prob(child_0(node)) ·(1 − P(X))+Prob(child_1(node))·P(X)
7:     end if
8: end function
```

Languages with non-binary choices such as Logic Programs with Annotated Disjunctions can be handled by encoding the choices with binary variables [21].

3 The DISPONTE Semantics for Probabilistic Ontologies

DISPONTE assigns a semantics to probabilistic ontologies following the approach of the distribution semantics for probabilistic logic programs. It defines a probability distribution over non-probabilistic ontologies called worlds. This probability distribution is extended to a joint distribution of the worlds and a query and the probability of the query is obtained by marginalization.

The probabilistic ontologies we consider associate to each axiom of the ontology a Boolean random variable that indicates whether the axiom is present in a world. A *probabilistic ontology* is thus a set of annotated axioms of the form

$$p_i :: A_i \tag{3}$$

or of unannotated axioms of the form A_i, for $i = 1, \ldots, n$, where p_i is the probability with which axiom A_i is included in a world. Let us call O_A the set $\{A_1, \ldots, A_n\}$ and X_i the Boolean random variable associated to axiom A_i. Each X_i is independent of every X_j with $i \neq j$. The probability of each X_i of being true is p_i. If the $p_i ::$ annotation is omitted for an axiom, we assume that the axiom is certain, i.e., that it has probability 1.

A *world* w is obtained by sampling a value for X_i for every axiom A_i of O_A and by including A_i in w if $X_i = 1$. Since the random variables for the different axioms are independent, the probability $P(w)$ of w is obtained as:

$$P(w) = \prod_{A_i \in w} p_i \prod_{A_j \in O_A \setminus w} (1 - p_j)$$

Given a query Q to O, we can define its conditional probability of being true given a world $P(Q|w)$ in the following intuitive way: $P(Q|w) = 1$ if $w \models Q$ and $P(Q|w) = 0$ if $w \not\models Q$.

The probability $P(Q)$ can be obtained from the joint distribution of the query and the worlds by the sum rule:

$$P(Q) = \sum_w P(Q, w) = \sum_w P(Q|w)P(w) = \sum_{w:w \models Q} P(w)$$

Similarly to the case of probabilistic logic programming, the probability of a query Q given a probabilistic ontology O can be computed by first finding the explanations for Q in O. An explanation in this context is a subset of axioms of O that is sufficient for entailing Q. Typically minimal explanations are sought for efficiency reasons. All the explanations for Q must be found, corresponding to all ways of proving Q. Let E_Q be set of explanations and e be an explanation from E_Q. The probability of Q can be obtained by computing the probability of the DNF formula

$$F(Q) = \bigvee_{e \in E_Q} \bigwedge_{A_i \in e} p_i$$

Example 2. This example is inspired by Examples 3.1, 4.1, 4.2 and 4.3 of [15] that describe a probabilistic ontology about cars. We know for sure that a $SportCar$ is a Car to which a max_speed greater than 245Km/h is associated:

$$SportsCar \sqsubseteq Car \sqcap \exists max\_speed.\, \geq_{245Km/h} \tag{4}$$

We also know that a Car is a subset of the class of vehicles $HasFourWheels$ with probability 0.9:

$$0.9 :: Car \sqsubseteq HasFourWheels \tag{5}$$

Please note that this does not mean that a member of the class Car is a member of $HasFourWheels$ with probability 0.9, see Section 5. $johns\_car$ is an instance of $SportsCar$ with probability 0.8:

$$0.8 :: johns\_car : SportsCar \tag{6}$$

We want to know what is the probability $P(Q_1)$ of axiom $Q_1 = johnsCar : HasFourWheels$ being true. Q_1 has a single explanation containing the axioms (4), (5) and (6). Since (4) is certain, $P(Q_1)$ is $0.8 \times 0.9 = 0.72$.

Example 3. Let us consider another example, inspired by the `people+pets` ontology proposed in [18]. We know that $kevin$ is a $DogOwner$ with probability 0.6 and a $CatOwner$ with probability 0.6:

$$0.6 :: kevin : DogOwner; \tag{7}$$

$$0.6 :: kevin : CatOwner. \tag{8}$$

Moreover we know for sure that *DogOwner* and *CatOwner* are subclasses of *PetOwner*

$$DogOwner \sqsubseteq PetOwner \tag{9}$$
$$CatOwner \sqsubseteq PetOwner \tag{10}$$

Then the query axiom $Q_2 = kevin : PetOwner$ has two explanations, one composed of the axioms (7) and (9) and the other composed of the axioms (8) and (10). Since (9) is certain, the probability of the first explanation is 0.6. Similarly, the probability of the second explanation is again 0.6. If we associate the Boolean random variable X_1 to (7) and X_2 to (8), the query axiom is true if the formula $X_1 \vee X_2$ is true. Thus, $P(Q_2) = P(X_1 \vee X_2)$. Since X_1 and X_2 are independent, we get $P(Q_2) = 0.6 + 0.6 - 0.6 \times 0.6 = 0.84$. As you can see, the fact that *kevin* is an instance of both *DogOwner* and *CatOwner* increases the probability that he is an instance of *PetOwner*: if he were an instance of *DogOwner* only, its probability of being a *PetOwner* would be 0.6 and similarly if he were an instance of *CatOwner* only.

Now suppose that we known that *PetOwner* is a subclass of *Ecologist* with probability 0.7:

$$0.7 :: PetOwner \sqsubseteq Ecologist \tag{11}$$

The query axiom $Q_3 = kevin : Ecologist$ has again two explanations, one composed of axioms (7), (9) and (11) and the other composed of the axioms (8), (10) and (11). Since (9) is certain, the probability of the first explanation is $0.4 \times 0.6 = 0.24$. Similarly, the probability of the second explanation is $0.5 \times 0.6 = 0.3$. If we associate the Boolean random variable X_3 to (11), Q_3 is a consequence of the theory if $X_1 \wedge X_3 \vee X_2 \wedge X_3$ is true. A BDD that can be built for this formula is the one shown in Figure 1 after replacing variable X_{21} with X_1, variable X_{22} with X_2 and variable X_{11} with X_3.

The probability of node n_3 computed by PROB is $0.7 \times 1 + 0.3 \times 0 = 0.7$. The probability of node n_2 is $0.6 \times 0.7 + 0.4 \times 0 = 0.42$ and the probability of node n_1 (and of Q_3) is $0.6 \times 0.7 + 0.4 \times 0.42 = 0.588$.

4 The BUNDLE System

BUNDLE computes the probability of a query Q given a probabilistic ontology O that follows the DISPONTE semantics. BUNDLE exploits an underlying ontology reasoner that is able to return all explanations for a query. One of these system is Pellet [23] that is a complete OWL-DL reasoner. Pellet takes as input an OWL ontology in various formats, including the RDFXML language.

In order to assign probabilities to axioms, we exploit the possibility given by OWL1.1 of declaring an annotation property for axioms. We thus annotate the axioms with the XML tag `bundle:probability` whose value should be a real number in [0,1].

BUNDLE takes as input two RDFXML files, one containing the ontology and one containing the annotations. For Example 3, the ontology file contains the following definition of $PetOwner$:

```
<owl:Class rdf:about="#PetOwner">
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Ecologist" />
    </rdfs:subClassOf>
</owl:Class>
```

The annotation file contains the annotation for the above axiom in the following form:

```
<owl11:Axiom>
    <rdf:subject rdf:resource="#PetOwner"/>
    <rdf:predicate rdf:resource="&rdfs;subClassOf"/>
    <rdf:object rdf:resource="#Ecologist"/>
    <bundle:probability>0.6</bundle:probability>
</owl11:Axiom>
```

BUNDLE first uses the annotation file for building a data structure $PMap$ that associates axioms with their probability. In order to do so, axioms are first converted to strings. We use the Manchester syntax to obtain a string representation of an axiom.

Then BUNDLE uses the EXPLAIN function of Pellet to compute explanations for a query axiom. BUNDLE thus accepts all the forms of query axioms that are accepted by Pellet's EXPLAIN function, namely subclass, instance, property value, theory inconsistency and class unsatisfiability.

Pellet returns the explanations for the query in the form of a set of sets of axioms. Then BUNDLE performs a double loop over the set of explanations and over the set of axioms in each explanation in which it builds a BDD representing the set of explanations. To manipulate BDDs we used the JavaBDD library[1] that provides a Java interface to the major BDD libraries such as CUDD[2].

Outside the outer loop, two data structures are initialized: $VarAxAnn$ is an array that maintains the association between Boolean random variables (whose index is the array index) and axioms together with their probability, and BDD represents the set of explanations. BDD is initialized to the BDD representing the zero Boolean function. Then the outer loop is entered in which $BDDE$ is initialized to the BDD representing the one Boolean function. In the inner loop the axioms of an explanation are considered one by one. Each axiom is first looked up in $PMap$ to get its probability. If NULL is returned this means that this is a certain axiom and it does not need to be considered anymore. Then the axiom is searched for in $VarAxAnn$ to see if it has already been assigned a random variable. If not, a cell is added to $VarAxAnn$ to store the axiom with its probability. At this point we know the axiom's position i in $VarAxAnn$

[1] http://javabdd.sourceforge.net/
[2] http://vlsi.colorado.edu/~fabio/CUDD/

and so the index of its Boolean variable X_i. We obtain a BDD representing $X_i = 1$ and we conjoin it with $BDDE$. At the end of the inner loop the BDD for the current explanation, $BDDE$, is disjoined with BDD. After the two cycles, function PROB of Figure 2 is called over BDD and its result is returned to the user.

BUNDLE has been implemented in Java and will be available for download from `http://sites.unife.it/bundle`. It has been successfully tested on various examples, including those of Section 3.

5 Discussion

The probabilistic knowledge that can be expressed with the DISPONTE semantics is epistemic by nature, namely it represents degrees of belief in the axioms rather that statistical information. While this is reasonable for many axioms, for subclass and subproperty axioms one may want to express statistical information, for example with a probabilistic subclass axiom $p :: A \sqsubseteq B$ one may want to express the fact that a random individual of A has probability p of belonging to B. The DISPONTE semantics, instead, interpret the axioms as stating that $A \sqsubseteq B$ is true with probability p. The difference is that, if two individuals i and j belong to class A, the probability that they both belong to B in the DISPONTE semantics is p while with a statistical interpretation is $p \times p$. Thus statistical information can be used to define a degree of partial overlap between classes. Extending DISPONTE to take account of this case is possible, it requires to define a probability distribution over models rather than over theories.

However, to reason with such knowledge, the inference engine must be modified in its inference procedure and cannot be used as a black box as in BUNDLE. In fact, BUNDLE assigns a single Boolean random variable to the axiom $A \sqsubseteq B$, while with a statistical interpretation a different Boolean random variable must be assigned to each assertion that an individual of class A belongs to class B. We leave this extension for future work.

Another limitation of BUNDLE is the use of the OWL 1.1 Axiom construct to specify probabilities. This seems to restrict the kind of axioms on which probabilities can be placed, since the object of the RDF triple does not allow complex class expressions. However this limitation can be overcome by defining a new class which is equivalent to the complex class expression and using the new class name in the RDF triple. In the future we plan to investigate the possibility of annotating the axioms directly in the ontology file.

As regards the complexity of reasoning on DISPONTE, it is equal to the complexity of the underlying description logic plus the #P complexity of computing the probability of a DNF formula of independent Boolean random variables, assuming the cost of keeping track of explanations during inference is negligible. Thus, the problem of inference in DISPONTE remains decidable if it was so in the underlying description logic.

6 Related Work

Our work differs from previous work in many respects. [6] proposed an extension of the description logic $\mathcal{ALC}$ that is able to express statistical information on the terminological knowledge such as partial concept overlapping. Similarly, [11] presents a probabilistic description logic based on Bayesian networks that deals with statistical terminological knowledge. As illustrated in Section 5, currently we are not able to express statistical terminological knowledge but it is possible to extend the semantics to do so. Differently from us, [6,11] do not allow probabilistic assertional knowledge about concept and role instances. [7] allows assertional knowledge about concept and role instances together with statistical terminological knowledge and combines the resulting probability distributions using cross-entropy minimization. In the future we plan to compare the DISPONTE semantics extended with statistical information with this approach.

[4] proposed a probabilistic extension of OWL that admits a translation into Bayesian networks. The semantics that is proposed assigns a probability distribution $P(i)$ over individuals, i.e. $\sum_i P(i) = 1$, and assigns a probability to a class C as $P(C) = \sum_{i \in C} P(i)$, while we assign a probability distribution over theories. PR-OWL [2,1] is an upper ontology that provides a framework for building probabilistic ontologies. It allows to use the first-order probabilistic logic MEBN [12] for representing uncertainty in ontologies and is thus very expressive but no implementation is available yet.

A different approach to the combination of description logic with probability is taken by [5,13,14] where the authors use probabilistic lexicographic entailment from probabilistic default reasoning. The logics proposed in these papers allow both terminological probabilistic knowledge as well as assertional probabilistic knowledge about instances of concepts and roles. PRONTO [9] is one of the systems that allows to perform inference in this semantics.

Similary to [7], the terminological knowledge is interpreted statistically while the assertional knowledge is interpreted epistemically by assigning degrees of beliefs to assertions, thus differing from our current treatment of terminological knowledge. Moreover it also allows to express default knowledge about concepts that can be overridden in subconcepts and whose semantics is given by Lehmann's lexicographic default entailment.

These works are based on Nilsson's probabilistic logic [16] where a probabilistic interpretation Pr defines a probability distribution over the set of interpretations $\mathcal{I}$. The probability of a logic formula ϕ according to Pr, denoted $Pr(\phi)$, is the sum of all $Pr(I)$ such that $I \in \mathcal{I}$ and $I \models \phi$.

A probabilistic knowledge base K is a set of probabilistic formulas of the form $\phi \geq p$. A probabilistic interpretation Pr satisfies $\phi \geq p$ iff $Pr(\phi) \geq p$. Pr satisfies K, or Pr is a model of K, iff Pr satisfies all $F \in K$. We say $\phi \geq p$ is a tight logical consequence of K iff p is the infimum of $Pr(\phi)$ subject to all models Pr of K. Computing tight logical consequences from probabilistic knowledge bases can be done by solving a linear optimization problem.

Nilsson's probabilistic logic differs from the distribution semantics: while a probabilistic knowledge base in Nilsson's logic may have multiple models that

are probabilistic interpretations, a probabilistic program under the distribution semantics has a single model that defines a single distribution over interpretations. Also, while in Nilsson's logic we want to compute the lowest p such that $Pr(\phi) \geq p$ holds for all Pr, in the distribution semantics we want to compute p such that $P(\phi) = p$. Nilsson's logic complexity is lower than the #P complexity of the distribution semantics.

In fact Nilsson's logic allows weaker conclusions than the distribution semantics. For example, consider a probabilistic program composed of $0.4 :: a.$ and $0.5 :: b.$ and a probabilistic knowledge base composed of $a \geq 0.4$ and $b \geq 0.5$. The distribution semantics allows to say that $P(a \vee b) = 0.7$, while with Nilsson's logic the lowest p such that $Pr(a \vee b) \geq p$ holds is 0.5. This is due to the fact that in the distribution semantics the probabilistic atoms are considered independent, which allows to make stronger conclusions. However, note that this does not restrict expressiveness as you can specify with the distribution semantics any joint probability distribution over the atoms of the Herbrand base interpreted as Boolean random variables, possibly introducing new random facts if needed.

Alternative approaches to modeling imperfect and incomplete knowledge in ontologies are based on fuzzy logic. A good survey of these approaches is presented in [15].

7 Conclusions

We have presented the semantics DISPONTE for probabilistic ontologies that is inspired by the distribution semantics of probabilistic logic programming. We have also presented the system BUNDLE that is able to compute the probability of queries from an uncertain OWL DL ontology.

In the future, we plan to extend DISPONTE to take into account statistical terminological knowledge and improve the way in which the input to BUNDLE is specified.

References

1. Carvalho, R.N., Laskey, K.B., Costa, P.C.: PR-OWL 2.0 - bridging the gap to OWL semantics. In: International Workshops on Uncertainty Reasoning for the Semantic Web (2010)
2. Costa, P.C.G., Laskey, K.B., Laskey, K.J.: Pr-owl: A bayesian ontology language for the semantic web. In: International Workshops on Uncertainty Reasoning for the Semantic Web. vol. 5327, pp. 88–107. Springer (2008)
3. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence. pp. 2462–2467 (2007)
4. Ding, Z., Peng, Y.: A probabilistic extension to ontology language OWL. In: Hawaii International Conference On System Sciences. IEEE (2004)
5. Giugno, R., Lukasiewicz, T.: P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web. In: European Conference on Logics in Artificial Intelligence. LNCS, vol. 2424, pp. 86–97. Springer (2002)

6. Heinsohn, J.: Probabilistic description logics. In: Conference on Uncertainty in Artificial Intelligence. pp. 311–318. Morgan Kaufmann (1994)
7. Jaeger, M.: Probabilistic reasoning in terminological logics. In: International Conference on Principles of Knowledge Representation and Reasoning. pp. 305–316 (1994)
8. Kimmig, A., Demoen, B., Raedt, L.D., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language problog. Theory Pract. Log. Program. 11(2-3), 235–262 (2011)
9. Klinov, P.: Pronto: A non-monotonic probabilistic description logic reasoner. In: European Semantic Web Conference. LNCS, vol. 5021, pp. 822–826. Springer (2008)
10. Koller, D.: Probabilistic relational models. In: International Workshop on Inductive Logic Programming. LNCS, vol. 1634, pp. 3–13. Springer (1999)
11. Koller, D., Levy, A.Y., Pfeffer, A.: P-classic: A tractable probablistic description logic. In: National Conference on Artificial Intelligence. pp. 390–397 (1997)
12. Laskey, K.B., da Costa, P.C.G.: Of starships and klingons: Bayesian logic for the 23rd century. In: Conference in Uncertainty in Artificial Intelligence. pp. 346–353. AUAI Press (2005)
13. Lukasiewicz, T.: Probabilistic default reasoning with conditional constraints. Ann. Math. Artif. Intell. 34(1-3), 35–88 (2002)
14. Lukasiewicz, T.: Expressive probabilistic description logics. Artif. Intell. 172(6-7), 852–883 (2008)
15. Lukasiewicz, T., Straccia, U.: Managing uncertainty and vagueness in description logics for the semantic web. J. Web Sem. 6(4), 291–308 (2008)
16. Nilsson, N.J.: Probabilistic logic. Artif. Intell. 28(1), 71–87 (1986)
17. Obrst, L., McCandless, D., Stoutenburg, S., Fox, K., Nichols, D., Prausa, M., Sward, R.: Evolving use of distributed semantics to achieve net-centricity. In: AAAI Fall Symposium (2007)
18. Patel-Schneider, P.F., Horrocks, I., Bechhofer, S.: Tutorial on OWL (2003), `http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/`
19. Poole, D.: Abducing through negation as failure: stable models within the independent choice logic. J. of Log. Program. 44(1-3), 5–35 (2000)
20. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
21. Riguzzi, F.: Extended semantics and inference for the Independent Choice Logic. Log. J. IGPL 17(6), 589–629 (2009)
22. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
23. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. 5(2), 51–53 (2007)
24. URW3-XG: Uncertainty reasoning for the World Wide Web, final report, `http://www.w3.org/2005/Incubator/urw3/XGR-urw3/`
25. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comp. 8(3), 410–421 (1979)
26. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: International Conference on Logic Programming. LNCS, vol. 3131, pp. 195–209. Springer (2004)