Undefined 0 (0) 1–26
IOS Press

Probabilistic Description Logics under the Distribution Semantics

Editor(s): Wolfgang Faber, University of Calabria, Italy, and University of Huddersfield, UK; Domenico Lembo, Sapienza University of Rome, Italy
Solicited review(s): Claudia d'Amato, University of Bari, Italy; Umberto Straccia, ISTI-CNR, Italy

Fabrizio Riguzzi [a] Elena Bellodi [b] Evelina Lamma [b] Riccardo Zese [b]
[a] *Dipartimento di Matematica e Informatica, University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy*
E-mail: fabrizio.riguzzi@unife.it
[b] *Dipartimento di Ingegneria, University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy*
E-mail: {elena.bellodi,evelina.lamma,riccardo.zese}@unife.it

Abstract. Representing uncertain information is crucial for modeling real world domains. In this paper we present a technique for the integration of probabilistic information in Description Logics (DLs) that is based on the distribution semantics for probabilistic logic programs. In the resulting approach, that we called DISPONTE, the axioms of a probabilistic knowledge base (KB) can be annotated with a real number between 0 and 1. A probabilistic knowledge base then defines a probability distribution over regular KBs called worlds and the probability of a given query can be obtained from the joint distribution of the worlds and the query by marginalization. We present the algorithm BUNDLE for computing the probability of queries from DISPONTE KBs. The algorithm exploits an underlying DL reasoner, such as Pellet, that is able to return explanations for queries. The explanations are encoded in a Binary Decision Diagram from which the probability of the query is computed. The experimentation of BUNDLE shows that it can handle probabilistic KBs of realistic size.

Keywords: Probabilistic Ontologies, Probabilistic Description Logics, OWL, Probabilistic Logic Programming, Distribution Semantics

1. Introduction

Representing uncertain information is of foremost importance in order to effectively model real world domains. This has been fully recognized in the field of Artificial Intelligence where uncertainty has been the focus of much research since its beginnings. In particular, the integration of logic and probability allows to model complex domains with many entities interconnected by uncertain relationships. This problem has been investigated by various authors both in the general case of first order logic [43,23,6] and in the case of restricted logics, such as Description Logics (DLs) and Logic Programming (LP).

DLs are fragments of first order logic particularly useful for knowledge representation. They are at the basis of the Semantic Web. The Web Ontology Language (OWL) is a family of languages that are syntactic variants of various DLs. Many proposals have appeared on the combination of probability theory and DLs [24,25,36,40,41]. Probabilistic DLs play an important role in the Semantic Web where knowledge may come from different sources and may have different reliability.

In LP, the distribution semantics [65] has emerged as one of the most effective approaches for representing probabilistic information. It underlies many probabilistic logic programming languages such as Probabilistic Horn Abduction [48], PRISM [65,66], Independent Choice Logic [49],

Logic Programs with Annotated Disjunctions [75], ProbLog [17] and CP-logic [73]. A program in one of these languages defines a probability distribution over normal logic programs called *worlds*. This distribution is extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs. The languages following the distribution semantics differ in the way they define the distribution over logic programs but have the same expressive power: there are transformations with linear complexity that can convert each one into the others [74,16].

The distribution semantics was applied successfully in many domains [17,66,8] and various inference and learning algorithms are available for it [32,54,9].

In [7,57,58] we applied this approach to DLs obtaining DISPONTE for "DIstribution Semantics for Probabilistic ONTologiEs" (Spanish for "get ready"). The idea is to annotate axioms of a theory with a probability and assume that each axiom is independent of the others. A DISPONTE knowledge base (KB) defines a probability distribution over regular KBs (worlds) and the probability of a query is obtained from the joint probability of the worlds and the query. The DISPONTE semantics differs from previous proposals because it provides a unified framework for representing different types of probabilistic knowledge, from assertional to terminological knowledge. DISPONTE can be applied to any DL, here we present it for a prototypical expressive DL, $\mathcal{SHOIN}(\mathbf{D})$, that is the basis of OWL DL.

We also present the algorithm BUNDLE for "Binary decision diagrams for Uncertain reasoNing on Description Logic thEories" [59], that performs inference over DISPONTE DLs. BUNDLE exploits an underlying reasoner such as Pellet [69] that returns explanations for queries. BUNDLE uses the inference techniques developed for probabilistic logic programs under the distribution semantics, in particular Binary Decision Diagrams (BDDs), for computing the probability of queries from a covering set of explanations. BUNDLE first finds explanations for the query and then encodes them in a BDD from which the probability can be computed in time linear in the size of the diagram.

BUNDLE's worst case complexity is high since the number of explanations may grow exponentially while the computation of the probability through BDDs has #P-complexity in the number of explanations. Nevertheless, we applied BUNDLE to various real world datasets and we found that it is able to handle domains of significant size.

The present paper extends previous work [7,57, 58,59] in various ways. With respect to the most recent work [59], the DISPONTE semantics is described in more detail with several examples; inference under this semantics is illustrated according to different techniques; we discuss how a lower bound on the probability of a query can be computed, with the quality of the bound monotonically increasing as more time for inference is allowed; BUNDLE is presented in more detail together with an extended description of Pellet's functions and a discussion about its computational complexity. Moreover, we elaborated a detailed comparison with related work and performed an extensive experimental analysis of BUNDLE in order to investigate its performance in practice. In particular we present a more in-depth comparison with PRONTO, a system for inference in the probabilistic DL P-$\mathcal{SHIQ}(\mathbf{D})$, and an analysis on the inference time trend with increasing sizes of the Tbox, the Abox and the set of probabilistic axioms.

The paper is organized as follows. Section 2 introduces Description Logics with particular reference to $\mathcal{SHOIN}(\mathbf{D})$ while Section 3 discusses DISPONTE. Section 4 illustrates how to compute the probability of queries to DISPONTE DLs and Section 5 describes the BUNDLE algorithm. Section 6 discusses the complexity of reasoning and Section 7 related work. Section 8 shows the results of experiments with BUNDLE and, finally, Section 9 concludes the paper.

2. Description Logics

Description Logics are knowledge representation formalisms that possess nice computational properties such as decidability and/or low complexity, see [1,2] for excellent introductions. DLs are particularly useful for representing ontologies and have been adopted as the basis of the Semantic Web.

While DLs are a fragment of predicate logic, they are usually represented using a syntax based on concepts and roles. A concept corresponds to a set of individuals of the domain while a role cor-

responds to a set of couples of individuals of the domain. In order to illustrate DLs, we describe $\mathcal{SHOIN}(\mathbf{D})$ as a prototype of expressive description logics. In the rest of the paper we use $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ to indicate *atomic concepts*, *atomic roles* and *individuals*, respectively. A *role* is either an atomic role $R \in \mathbf{R}$ or the inverse R^- of an atomic role $R \in \mathbf{R}$. We use $\mathbf{R}^-$ to denote the set of all inverses of roles in $\mathbf{R}$. *Concepts* are defined as follows. Each $A \in \mathbf{A}$, $\bot$ and $\top$ are concepts and if $a \in \mathbf{I}$, then $\{a\}$ is a concept called a *nominal*. If C, C_1 and C_2 are concepts and $R \in \mathbf{R} \cup \mathbf{R}^-$, then $(C_1 \sqcap C_2)$, $(C_1 \sqcup C_2)$ and $\neg C$ are concepts, as well as $\exists R.C$ and $\forall R.C$ and $\geq nR$ and $\leq nR$ for an integer $n \geq 0$.

An *RBox* $\mathcal{R}$ consists of a finite set of *transitivity axioms* $Trans(R)$, where $R \in \mathbf{R}$, and *role inclusion axioms* $R \sqsubseteq S$, where $R, S \in \mathbf{R} \cup \mathbf{R}^-$. A *TBox* $\mathcal{T}$ is a finite set of *concept inclusion axioms* $C \sqsubseteq D$, where C and D are concepts. We use $C \equiv D$ to abbreviate $C \sqsubseteq D$ and $D \sqsubseteq C$. An *ABox* $\mathcal{A}$ is a finite set of *concept membership axioms* $a : C$, *role membership axioms* $(a, b) : R$, *equality axioms* $a = b$ and *inequality axioms* $a \neq b$, where C is a concept, $R \in \mathbf{R}$ and $a, b \in \mathbf{I}$. A *knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$ and an ABox $\mathcal{A}$.

A $\mathcal{SHOIN}(\mathbf{D})$ KB $\mathcal{K}$ is assigned a semantics in terms of interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta^\mathcal{I}$ is a non-empty *domain* and $\cdot^\mathcal{I}$ is the *interpretation function* that assigns an element in $\Delta^\mathcal{I}$ to each $a \in \mathbf{I}$, a subset of $\Delta^\mathcal{I}$ to each $A \in \mathbf{A}$ and a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ to each $R \in \mathbf{R}$. The mapping $\cdot^\mathcal{I}$ is extended to all concepts (where $R^\mathcal{I}(x) = \{y | (x, y) \in R^\mathcal{I}\}$ and $\#X$ denotes the cardinality of the set X) as:

$$
\begin{aligned}
(R^-)^\mathcal{I} &= \{(y, x) | (x, y) \in R^\mathcal{I}\} \\
\top^\mathcal{I} &= \Delta^\mathcal{I} \\
\bot^\mathcal{I} &= \emptyset \\
\{a\}^\mathcal{I} &= \{a^\mathcal{I}\} \\
(C_1 \sqcap C_2)^\mathcal{I} &= C_1^\mathcal{I} \cap C_2^\mathcal{I} \\
(C_1 \sqcup C_2)^\mathcal{I} &= C_1^\mathcal{I} \cup C_2^\mathcal{I} \\
(\neg C)^\mathcal{I} &= \Delta^\mathcal{I} \setminus C^\mathcal{I} \\
(\forall R.C)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} | R^\mathcal{I}(x) \subseteq C^\mathcal{I}\} \\
(\exists R.C)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} | R^\mathcal{I}(x) \cap C^\mathcal{I} \neq \emptyset\} \\
(\geq nR)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} | \#R^\mathcal{I}(x) \geq n\} \\
(\leq nR)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} | \#R^\mathcal{I}(x) \leq n\}
\end{aligned}
$$

$\mathcal{SHOIN}(\mathbf{D})$ adds to $\mathcal{SHOIN}$ datatype roles, i.e., roles that map an individual to an element of a datatype such as integers, floats, etc. Then new concept definitions involving datatype roles are added that mirror those involving roles introduced above. We also assume that we have predicates over the datatypes.

The *satisfaction* of an axiom E in an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, denoted by $\mathcal{I} \models E$, is defined as follows: (1) $\mathcal{I} \models Trans(R)$ iff $R^\mathcal{I}$ is transitive, (2) $\mathcal{I} \models R \sqsubseteq S$ iff $R^\mathcal{I} \subseteq S^\mathcal{I}$, (3) $\mathcal{I} \models C \sqsubseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$, (4) $\mathcal{I} \models a : C$ iff $a^\mathcal{I} \in C^\mathcal{I}$, (5) $\mathcal{I} \models (a, b) : R$ iff $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$, (6) $\mathcal{I} \models a = b$ iff $a^\mathcal{I} = b^\mathcal{I}$, (7) $\mathcal{I} \models a \neq b$ iff $a^\mathcal{I} \neq b^\mathcal{I}$. $\mathcal{I}$ *satisfies* a set of axioms $\mathcal{E}$, denoted by $\mathcal{I} \models \mathcal{E}$, iff $\mathcal{I} \models E$ for all $E \in \mathcal{E}$. An interpretation $\mathcal{I}$ *satisfies* a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$, denoted $\mathcal{I} \models \mathcal{K}$, iff $\mathcal{I}$ satisfies $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{A}$. In this case we say that $\mathcal{I}$ is a *model* of $\mathcal{K}$.

A knowledge base $\mathcal{K}$ is *satisfiable* iff it has a model. An axiom E is *entailed* by $\mathcal{K}$, denoted $\mathcal{K} \models E$, iff every model of $\mathcal{K}$ satisfies also E. A concept C is *satisfiable* relative to $\mathcal{K}$ iff $\mathcal{K}$ has a model $\mathcal{I}$ such that $C^\mathcal{I} \neq \emptyset$.

A DL is *decidable* if the problem of checking the satisfiability of a KB is decidable. In particular, $\mathcal{SHOIN}(\mathbf{D})$ is decidable iff there are no number restrictions on non-simple roles. A role is *non-simple* iff it is transitive or it has transitive subroles.

A query over a KB is usually an axiom for which we want to test the entailment from the KB. The entailment test may be reduced to checking the unsatisfiability of a concept in the KB, i.e., the emptiness of the concept. For example, the entailment of the axiom $C \sqsubseteq D$ may be tested by checking the unsatisfiability of the concept $C \sqcap \neg D$.

3. The DISPONTE Semantics for DLs

DISPONTE applies the distribution semantics [65] of probabilistic logic programming to DLs. A program following this semantics defines a probability distribution over normal logic programs called *worlds*. Then the distribution is extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs.

3.1. Syntax and Intuition

In DISPONTE, a *probabilistic knowledge base* $\mathcal{K}$ is a set of certain axioms or probabilistic axioms. *Certain axioms* take the form of regular DL axioms. *Probabilistic axioms* take the form

$$p :: E \tag{1}$$

where p is a real number in $[0, 1]$ and E is a DL axiom.

The idea of DISPONTE is to associate independent Boolean random variables to the probabilistic axioms. By assigning values to every random variable we obtain a *world*, the set of axioms whose random variables are assigned the value 1.

Every formula obtained from a certain axiom is included in a world w. For each probabilistic axiom, we decide whether to include it or not in w. A world therefore is a non probabilistic KB that can be assigned a semantics in the usual way. A query is entailed by a world if it is true in every model of the world.

The probability p can be interpreted as an *epistemic probability*, i.e., as the degree of our belief in axiom E. For example, a probabilistic concept membership axiom

$$p :: a : C$$

means that we have degree of belief p in $C(a)$. The statement that Tweety flies with probability 0.9 can be expressed as

$$0.9 :: tweety : Flies$$

A probabilistic concept inclusion axiom of the form

$$p :: C \sqsubseteq D \tag{2}$$

represents the fact that we believe in the truth of $C \sqsubseteq D$ with probability p. For example, the axiom

$$0.9 :: Bird \sqsubseteq Flies \tag{3}$$

means that birds fly with a 90% probability.

3.2. Semantics

We follow the approach of [50] and first give some definitions. An *atomic choice* is a couple (E_i, k) where E_i is the ith probabilistic axiom and $k \in \{0, 1\}$. k indicates whether E_i is chosen to be included in a world ($k = 1$) or not ($k = 0$). A set of atomic choices κ is *consistent* if only one decision is taken for each probabilistic axiom; we assume independence between the different choices. A *composite choice* κ is a consistent set of atomic choices. The probability of a composite choice κ is $P(\kappa) = \prod_{(E_i,1)\in\kappa} p_i \prod_{(E_i,0)\in\kappa}(1-p_i)$, where p_i is the probability associated with axiom E_i. A *selection* σ is a total composite choice, i.e., it contains an atomic choice (E_i, k) for every probabilistic axiom of the theory. A selection σ identifies a theory w_σ called a *world* in this way: $w_\sigma = \mathcal{C} \cup \{E_i | (E_i, 1) \in \sigma\}$ where $\mathcal{C}$ is the set of certain axioms. Let us indicate with $\mathcal{S}_\mathcal{K}$ the set of all selections and with $\mathcal{W}_\mathcal{K}$ the set of all worlds. The probability of a world w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(E_i,1)\in\sigma} p_i \prod_{(E_i,0)\in\sigma}(1 - p_i)$. $P(w_\sigma)$ is a probability distribution over worlds, i.e., $\sum_{w\in\mathcal{W}_\mathcal{K}} P(w) = 1$.

We can now assign probabilities to queries. Given a world w, the probability of a query Q is defined as $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of a query can be obtained by marginalizing the joint probability of the query and the worlds:

$$P(Q) = \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q, w) \tag{4}$$

$$= \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q|w)P(w) \tag{5}$$

$$= \sum_{w\in\mathcal{W}_\mathcal{K}: w\models Q} P(w) \tag{6}$$

where (4) and (5) follow for the sum and product rule of the theory of probability respectively and (6) holds because $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise.

Example 1. *Consider the following KB, inspired by the* `people+pets` *ontology proposed in [44]:*

$$0.5 :: \exists hasAnimal.Pet \sqsubseteq NatureLover \quad (7)$$

$$(kevin, fluffy) : hasAnimal \quad (8)$$

$$(kevin, tom) : hasAnimal \quad (9)$$

$$fluffy : Cat \quad (10)$$

$$tom : Cat \quad (11)$$

$$0.6 :: Cat \sqsubseteq Pet \quad (12)$$

The KB indicates that the individuals that own an animal which is a pet are nature lovers with a 50% probability and that kevin has the animals fluffy and tom. Fluffy and tom are cats and cats are pets with probability 60%. The KB has four possible worlds, corresponding to the selections:

$$\{((7), 1), ((12), 1)\}$$
$$\{((7), 1), ((12), 0)\}$$
$$\{((7), 0), ((12), 1)\}$$
$$\{((7), 0), ((12), 0)\}$$

and the query axiom $Q = kevin : NatureLover$ is true in the first of them, while in the remaining ones it is false. Each pair in the selections contains the axiom identifier and the value of its selector (k).

The probability of the query is $P(Q) = 0.5 \cdot 0.6 = 0.3$.

Example 2. *Let us consider a slightly different knowledge base:*

$$\exists hasAnimal.Pet \sqsubseteq NatureLover$$

$$(kevin, fluffy) : hasAnimal$$

$$(kevin, tom) : hasAnimal$$

$$0.4 :: fluffy : Cat \quad (13)$$

$$0.3 :: tom : Cat \quad (14)$$

$$0.6 :: Cat \sqsubseteq Pet \quad (15)$$

Here individuals that own an animal which is a pet are surely nature lovers and kevin has the animals fluffy and tom. Moreover, we believe in the fact that fluffy and tom are cats and that cats are pets with a certain probability.

This KB has eight worlds and the query axiom $Q = kevin : NatureLover$ is true in three of them, those corresponding to the following selections:

$$\{((13), 1), ((14), 0), ((15), 1)\}$$
$$\{((13), 0), ((14), 1), ((15), 1)\}$$
$$\{((13), 1), ((14), 1), ((15), 1)\}$$

so the probability is

$$P(Q) = 0.4 \cdot 0.7 \cdot 0.6 + 0.6 \cdot 0.3 \cdot 0.6 + 0.4 \cdot 0.3 \cdot 0.6 = 0.348.$$

Note that if the regular DL KB obtained by stripping the probabilistic annotations is inconsistent, there will be worlds that are inconsistent too. These worlds will entail the query trivially, as does the regular KB. A DISPONTE KB with inconsistent worlds should not be used to derive consequences, just as a regular DL KB that is inconsistent should not.

However, apparently contradictory probabilistic information is allowed. For example, the KB

$$0.9 :: Bird \sqsubseteq Flies$$

$$0.1 :: tweety : Flies$$

$$tweety : Bird$$

states that the probability of flying for a bird is 0.9 and the probability of flying for *tweety*, a particular bird, is 0.1. The two probabilistic statements are considered as independent evidence for *tweety* flying and are combined giving the probability 0.91 for the query $tweety : Flies$. In fact, this KB has four worlds and $tweety : Flies$ is true in three of them, giving $P(Q) = 0.9 \cdot 0.1 + 0.9 \cdot 0.9 + 0.1 \cdot 0.1 = 0.91$. Thus knowledge about instances of the domain may reinforce general knowledge and vice-versa.

Example 3. *The knowledge base*

$$kevin : \forall friend.Person$$

$$(kevin, laura) : friend$$

$$(laura, diana) : friend$$

$$0.4 :: Trans(friend)$$

means that all individuals in the friend relationship with kevin are persons, that kevin is a friend of laura, that laura is a friend of diana and that

given three individuals a, b and c, there is a 40% probability that if a is a friend of b and b is a friend of c then a is a friend of c. In particular, we have a 40% probability that, if kevin is a friend of laura and laura is a friend of diana, then kevin is a friend of diana. Since the first two are certain facts, then kevin is a friend of diana with a 40% probability and diana is a person also with a 40% probability.

The final report of the W3C Uncertainty Reasoning for the World Wide Web Incubator Group [71] discusses the challenge of reasoning with uncertain information on the World Wide Web. Moreover, it also highglights several use cases for the representation of uncertainty: combining knowledge from multiple, untrusted sources; discovering and using services in the presence of uncertain information on the user requirements and the service descriptions; recommending items to users; extracting and annotating information from the web; automatically performing tasks for users such as making an appointment, and handling healthcare and life sciences information and knowledge.

By introducing probability in an expressive description logic, such as $\mathcal{SHOIN}(\mathbf{D})$ that is one of the bases of OWL, we are able to tackle these problems as shown in the following example.

Example 4. *Consider a KB similar to the one of Example 2 but where we have a single cat, fluffy. Suppose the user has the knowledge*

$$\exists hasAnimal.Pet \sqsubseteq NatureLover$$
$$(kevin, fluffy) : hasAnimal$$
$$Cat \sqsubseteq Pet$$

and there are two sources of information with different reliability that provide the information that fluffy is a cat. On one source the user has a degree of belief of 0.4, i.e., he thinks it is correct with a 40% probability, while on the other source he has a degree of belief 0.3, i.e., he thinks it is correct with a 30% probability. The user can reason on this knowledge by adding the following statements to his KB:

$$0.4 \quad :: \quad fluffy : Cat \tag{16}$$
$$0.3 \quad :: \quad fluffy : Cat \tag{17}$$

The two statements represent independent evidence on fluffy being a cat.

The query axiom $Q = kevin : NatureLover$ is true in 3 out of the 4 worlds, those corresponding to the selections:

$$\{((16), 1), ((17), 1)\}$$
$$\{((16), 1), ((17), 0)\}$$
$$\{((16), 0), ((17), 1)\}$$

So $P(Q) = 0.4 \cdot 0.3 + 0.4 \cdot 0.7 + 0.6 \cdot 0.3 = 0.58$. This is reasonable if the two sources can be considered as independent. In fact, the probability comes from the disjunction of two independent Boolean random variables with probabilities respectively 0.4 and 0.3:

$$\begin{aligned} P(Q) &= P(X_1 \vee X_2) \\ &= P(X_1) + P(X_2) - P(X_1 \wedge X_2) \\ &= P(X_1) + P(X_2) - P(X_1)P(X_2) \\ &= 0.4 + 0.3 - 0.4 \cdot 0.3 = 0.58 \end{aligned}$$

4. Inference

We propose an approach for performing inference over DISPONTE DLs in which we first find explanations for the given query and then compute the probability of the query from them. In order to discuss the approach, we first need to introduce some definitions.

A composite choice κ *identifies* a set of worlds $\omega_\kappa = \{w_\sigma | \sigma \in \mathcal{S}_\mathcal{K}, \sigma \supseteq \kappa\}$, the set of worlds whose selection is a superset of κ, i.e., the set of worlds "compatible" with κ. We define the set of worlds *identified* by a set of composite choices K as $\omega_K = \bigcup_{\kappa \in K} \omega_\kappa$.

A composite choice κ is an *explanation* for a query Q if Q is entailed by every world of ω_κ. A *covering* set of explanations for Q is a set of composite choices K such that every world w_σ in which the query is true is included in ω_K.

Two composite choices κ_1 and κ_2 are *incompatible* if their union is inconsistent. For example, the composite choices $\kappa_1 = \{(E_i, 1)\}$ and $\kappa_2 = \{(E_i, 0)\}$ are incompatible where E_i is a probabilistic axiom. A set K of composite choices is *pairwise incompatible* if for all $\kappa_1 \in K, \kappa_2 \in$

$K, \kappa_1 \neq \kappa_2$ implies κ_1 and κ_2 are incompatible. For example

$$K = \{\kappa_1, \kappa_2\} \tag{18}$$

with

$$\kappa_1 = \{(E_i, 1)\}$$

and

$$\kappa_2 = \{(E_i, 0), (E_l, 1)\} \tag{19}$$

is pairwise incompatible.

We define *the probability of a pairwise incompatible set of composite choices* K as

$$P(K) = \sum_{\kappa \in K} P(\kappa) \tag{20}$$

Two sets of composite choices K_1 and K_2 are *equivalent* if $\omega_{K_1} = \omega_{K_2}$, i.e., if they identify the same set of worlds. For example, K in (18) is equivalent to

$$K' = \{\kappa'_1, \kappa'_2\} \tag{21}$$

with

$$\kappa'_1 = \{(E_i, 1)\}$$

and

$$\kappa'_2 = \{(E_l, 1)\} \tag{22}$$

4.1. Inference with the splitting algorithm

If E is an axiom and κ is a composite choice such that $\kappa \cap \{(E, 0), (E, 1)\} = \emptyset$, the *split* of κ on E is the set of composite choices $S_{\kappa,E} = \{\kappa \cup \{(E, 0)\}, \kappa \cup \{(E, 1)\}\}$. It is easy to see that κ and $S_{\kappa,E}$ identify the same set of possible worlds, i.e., that $\omega_\kappa = \omega_{S_{\kappa,E}}$. For example, the split of κ'_2 in (22) on E_i contains κ_2 in (19) and $\{(E_i, 1), (E_l, 1)\}$.

Poole [50] proved the following result.

Theorem 1. *Given a finite set K of finite composite choices, there exists a finite set K' of pairwise incompatible finite composite choices such that K and K' are equivalent.*

Proof. Given a finite set of finite composite choices K, there are two possibilities to form a new set K' of composite choices so that K and K' are equivalent:

1. **removing dominated elements**: if $\kappa_1, \kappa_2 \in K$ and $\kappa_1 \subset \kappa_2$, let $K' = K \setminus \{\kappa_2\}$.
2. **splitting elements**: if $\kappa_1, \kappa_2 \in K$ are compatible (and neither is a superset of the other), there is a $(E, k) \in \kappa_1 \setminus \kappa_2$. We replace κ_2 by the split of κ_2 on E. Let $K' = K \setminus \{\kappa_2\} \cup S_{\kappa_2,E}$.

In both cases $\omega_K = \omega_{K'}$. If we repeat these two operations until neither is applicable we obtain a splitting algorithm (see Figure 1) that terminates because K is a finite set of finite composite choices. The resulting set K' is pairwise incompatible and is equivalent to the original set. For example, the splitting algorithm applied to K' (21) can return K (18). □

Theorem 2 ([48])**.** *If K_1 and K_2 are both pairwise incompatible finite sets of finite composite choices such that they are equivalent then $P(K_1) = P(K_2)$.*

For example, K in (18) and $K'' = \{\kappa''_1, \kappa''_2\}$ with $\kappa''_1 = \{(E_i, 1), (E_l, 0)\}$ and $\kappa''_2 = \{(E_l, 1)\}$ are equivalent and are both pairwise incompatible. Their probabilities are

$$P(K) = p_i + (1 - p_i)p_l = p_i + p_l - p_i p_l$$

and

$$P(K'') = p_i(1 - p_l) + p_l = p_i + p_l - p_i p_l$$

Note that if we compute the probability of K' in (21) with formula (20) we would obtain $p_i + p_l$ which is different from the probabilities of K and K'' above, even if K' is equivalent to K and K'', because K' is not pairwise incompatible.

We can thus define the probability $P(K)$ of a generic set of composite choices K as $P(K) = P(K')$, where K' is a mutually incompatible set of composite choices that is equivalent to K, i.e., such that $\omega_{K'} = \omega_K$. Given a query Q, the set $K_Q = \{\sigma | \sigma \in \mathcal{S}_\mathcal{K} \wedge w_\sigma \models Q\}$ is a set of pairwise incompatible composite choices. Since $P(Q) = \sum_{\sigma \in K_Q} P(\sigma)$, then $P(Q) = P(K_Q)$.

If K' is a set of explanations for Q that is covering, then K' and K_Q are equivalent so $P(Q) =$

```
1: procedure SPLIT(K)
2:     Input: set of composite choices K
3:     Output: pairwise incompatible set of composite choices equivalent to K
4:     loop
5:         if ∃κ1, κ2 ∈ K and κ1 ⊂ κ2 then
6:             K ← K \ {κ2}
7:         else
8:             if ∃κ1, κ2 ∈ K compatible then
9:                 choose (E, k) ∈ κ1 \ κ2
10:                K ← K \ {κ2} ∪ S_{κ2,E}
11:            else
12:                exit and return K
13:            end if
14:        end if
15:    end loop
16: end procedure
```

Fig. 1. Splitting Algorithm.

$P(K_Q) = P(K')$. Thus we do not have to generate all worlds where a query is true in order to compute its probability, finding a mutually incompatible covering set of explanations is enough. Moreover, an additional result is given by the following theorem.

Theorem 3. *Given two finite sets of finite composite choices K_1 and K_2, if $K_1 \subseteq K_2$, then $P(K_1) \leq P(K_2)$.*

Proof. Let K_1' be the result of the application of the splitting algorithm to K_1. We can apply the same operations of the algorithm to K_2 obtaining $K = K_1' \cup K'$ for a certain K'. At this point, we can continue applying the splitting algorithm. If there exists a $\kappa \in K$ and a $\kappa' \in K$ such that $\kappa' \subseteq \kappa$, at least one of κ and κ' must not belong to K_1', otherwise κ would have been removed when splitting K_1'. κ is removed from K while κ' remains. If there is a compatible couple κ_1 and κ_2 in K, we can assume that one of the two, say κ_2, does not belong to K_1', since otherwise it would have been split before. We add to K the split of κ_2 on an atomic choice in κ_1 but not in κ_2.

Let K_2' be the results of the splitting algorithm so applied. K_2' is such that, for each element κ_1 of K_1', K_2' contains an element κ_2 such that $\kappa_2 \subseteq \kappa_1$. Therefore, in the summation in (20), for each term in $P(K_1')$ there will be a term in $P(K_2')$ with a larger or equal value so $P(K_1') \leq P(K_2')$. □

Thus, if K is a finite set of finite explanations for a query Q but we don't know if K contains all possible explanations for Q, i.e., we don't know whether K is covering, then $P(K)$ will be a lower bound of $P(Q)$. So we can compute progressively more accurate estimates from below of $P(Q)$ by considering an increasing set of explanations. Only when K contains all possible explanations, then $P(K) = P(Q)$.

The problem of computing the probability of a query can thus be reduced to that of finding a covering set of explanations K and then making it pairwise incompatible, so that the probability can be computed with the summation of (20). To obtain a pairwise incompatible set of explanations, the splitting algorithm can be applied.

4.2. Inference through Binary Decision Diagrams

As an alternative to the splitting algorithm, given a covering set of explanations K for a query Q, we can define the Disjunctive Normal Form (DNF) Boolean formula f_K as

$$f_K(\mathbf{X}) = \bigvee_{\kappa \in K} \bigwedge_{(E_i,1) \in \kappa} X_i \bigwedge_{(E_i,0) \in \kappa} \overline{X_i} \tag{23}$$

The variables $\mathbf{X} = \{X_i | (E_i, k) \in \kappa, \kappa \in K\}$ are independent Boolean random variables whose probability of being true is p_i for the variable X_i. The probability that $f_K(\mathbf{X})$ assumes value 1 is equal to the probability of Q. We can now apply *knowledge compilation* to the propositional formula $f_K(\mathbf{X})$ [15], i.e. translate it to a target language that allows answering queries in polynomial time. A target language that was found to give good performances is the one of Binary Decision Diagrams (BDD). From a BDD we can compute the probability of the query with a dynamic programming algorithm that is linear in the size of the BDD [17]. Riguzzi [54] showed that this approach is faster than the splitting algorithm.

A BDD for a function of Boolean variables is a rooted graph that has one level for each Boolean variable. A node n in a BDD has two children:

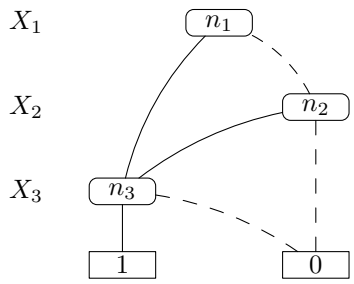


Fig. 2. BDD for function (24).

one corresponding to the 1 value of the variable associated with the level of n, indicated with $child_1(n)$, and one corresponding to the 0 value of the variable, indicated with $child_0(n)$. When drawing BDDs, the 0-branch - the one going to $child_0(n)$ - is distinguished from the 1-branch by drawing it with a dashed line. The leaves store either 0 or 1.

BDDs can be built by combining simpler BDDs using Boolean operators. While building BDDs, simplification operations can be applied that delete or merge nodes. Merging is performed when the diagram contains two identical sub-diagrams, while deletion is performed when both arcs from a node point to the same node. In this way a reduced BDD is obtained, often with a much smaller number of nodes with respect to the original BDD. The size of the reduced BDD depends on the order of the variables: finding an optimal order is an NP-complete problem [10] and several heuristic techniques are used in practice by highly efficient software packages such as CUDD[1]. Alternative methods involve learning variable order from examples [21].

For instance, a BDD for the function

$$f(\mathbf{X}) = (X_1 \wedge X_3) \vee (X_2 \wedge X_3) \tag{24}$$

is shown in Figure 2. A BDD performs a Shannon expansion of the Boolean formula $f_K(\mathbf{X})$, so that, if X is the variable associated with the root level of a BDD, the formula $f_K(\mathbf{X})$ can be represented as $f_K(\mathbf{X}) = X \wedge f_K^X(\mathbf{X}) \vee \overline{X} \wedge f_K^{\overline{X}}(\mathbf{X})$ where $f_K^X(\mathbf{X})$ ($f_K^{\overline{X}}(\mathbf{X})$) is the formula obtained by $f_K(\mathbf{X})$ by setting X to 1 (0). Now the two disjuncts are pairwise exclusive and the probability of $f_K(\mathbf{X})$ can be computed as $P(f_K(\mathbf{X})) = P(X)P(f_K^X(\mathbf{X})) + (1 - P(X))P(f_K^{\overline{X}}(\mathbf{X}))$. In other words, BDDs make the explanations pairwise incompatible. Figure 3 shows function Prob that implements the dynamic programming algorithm for computing the probability of a formula encoded as a BDD.

```
1: function Prob(node)
2:     Input: a BDD node
3:     Output: the probability of the Boolean function associated with the node
4:     if node is a terminal then
5:         return value(node)
6:     else
7:         let X be v(node)
8:         P_1 ←Prob(child_1(node))
9:         P_0 ←Prob(child_0(node))
10:        return P(X) · P_1 + (1 − P(X)) · P_0
11:    end if
12: end function
```

Fig. 3. Function that computes the probability of a formula encoded as a BDD, where $value(node)$ is 0 or 1 and $v(node)$ is the variable associated with $node$.

The function should also store the value of already visited nodes in a table so that, if a node is visited again, its probability can be retrieved from the table. For the sake of simplicity Figure 3 does not show this optimization but it is fundamental to achieve linear cost in the number of nodes, as without it the cost of the function Prob would be proportional to 2^n where n is the number of Boolean variables.

Let us discuss inference on some examples.

Example 5. *Let us consider the KB of Example 2. A covering set of explanations for the query axiom Q = kevin : NatureLover is $K = \{\kappa_1, \kappa_2\}$ where $\kappa_1 = \{((13), 1), ((15), 1)\}$ and $\kappa_2 = \{((14), 1), ((15), 1)\}$. If we associate the random variables X_1 to (13), X_2 to (14) and X_3 to (15), $f_K(\mathbf{X})$ is shown in (24) and the BDD associated with the set K of explanations is shown in Figure 2. By applying the algorithm in Figure 3 we get*

$$\text{Prob}(n_3) = 0.6 \cdot 1 + 0.4 \cdot 0 = 0.6$$

$$\text{Prob}(n_2) = 0.4 \cdot 0.6 + 0.6 \cdot 0 = 0.24$$

$$\text{Prob}(n_1) = 0.3 \cdot 0.6 + 0.7 \cdot 0.24 = 0.348$$

so $P(Q) = \text{Prob}(n_1) = 0.348$ which corresponds to the probability given by the semantics.

[1] Available at `http://vlsi.colorado.edu/~fabio/CUDD/`

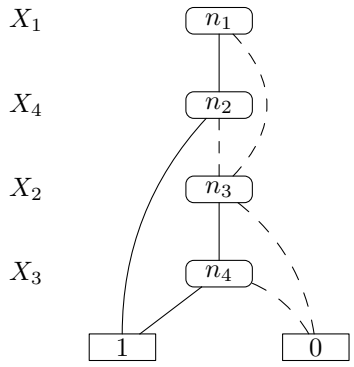


Fig. 4. BDD for Example 6.

Example 6. *Let us consider a slightly different knowledge base:*

$$\exists hasAnimal.Pet \sqsubseteq NatureLover$$

$$(kevin, fluffy) : hasAnimal$$

$$(kevin, tom) : hasAnimal$$

$$0.4 :: fluffy : Dog \quad (25)$$

$$0.3 :: tom : Cat \quad (26)$$

$$0.6 :: Cat \sqsubseteq Pet \quad (27)$$

$$0.5 :: Dog \sqsubseteq Pet \quad (28)$$

A covering set of explanations for the query axiom $Q = kevin : NatureLover$ *is* $K = \{\kappa_1, \kappa_2\}$ *where* $\kappa_1 = \{((25), 1), ((28), 1)\}$ *and* $\kappa_2 = \{((26), 1), ((27), 1)\}$*. If we associate the random variables* X_1 *to (25),* X_2 *to (26),* X_3 *to (27) and* X_4 *to (28), the BDD associated with the set* K *of explanations is shown in Figure 4.*
By applying the algorithm in Figure 3 we get

$$\text{Prob}(n_4) = 0.6 \cdot 1 + 0.4 \cdot 0 = 0.6$$

$$\text{Prob}(n_3) = 0.3 \cdot 0.6 + 0.7 \cdot 0 = 0.18$$

$$\text{Prob}(n_2) = 0.5 \cdot 1 + 0.5 \cdot 0.18 = 0.59$$

$$\text{Prob}(n_1) = 0.4 \cdot 0.59 + 0.6 \cdot 0.18 = 0.344$$

so $P(Q) = \text{Prob}(n_1) = 0.344$.

5. BUNDLE

The BUNDLE algorithm computes the probability of queries from a probabilistic knowledge base that follows the DISPONTE semantics. It first finds a covering set of explanations for the query and then makes them pairwise incompatible by using BDDs. Finally, it computes the probability from the BDD by using function Prob in Figure 3.

The problem of finding explanations for a query has been investigated by various authors [28,31,22, 30]. Schlobach and Cornet [67] called it *axiom pinpointing* and considered it a non-standard reasoning service useful for tracing derivations and debugging ontologies. In particular, they define *minimal axiom sets* or *MinAs* for short.

Definition 1 (MinAs)**.** *Let* $\mathcal{K}$ *be a KB and* Q *an axiom that follows from it, i.e.,* $\mathcal{K} \models Q$*. We call a set* $M \subseteq \mathcal{K}$ *a* minimal axiom set *or* MinA *for* Q *in* $\mathcal{K}$ *if* $M \models Q$ *and* M *is minimal w.r.t. set inclusion. The set of all possible MinAs is called* All-MinAs$(Q, \mathcal{K})$*. The problem of enumerating all MinAs is called* min-a-enum*.*

All-MinAs$(Q, \mathcal{K})$ can be used to derive a covering set of explanations. min-a-enum can be solved either with reasoner dependent (glass-box) approaches or reasoner independent (black-box) approaches [30]. Glass-box approaches are built on existing tableau-based decision procedures and modify the internals of the reasoner. Black-box approaches use the DL reasoner solely as a subroutine and the internals of the reasoner do not need to be modified.

The techniques of [29,31,22,30] for axiom pinpointing have been integrated into the Pellet reasoner [69]. By default, Pellet solves min-a-enum with a hybrid glass/black-box approach: it finds a single MinA using a modified tableau algorithm and then finds all the other MinAs using a black box method (the *hitting set tree* algorithm). The method involves removing an axiom of the MinA from the KB and looking for alternative explanations. By repeating this process until the query is not entailed, the set of all explanations is found.

BUNDLE is based on Pellet and uses it for solving the min-a-enum problem. In the following, we first illustrate Pellet's algorithm for solving min-a-enum and then we show the whole BUNDLE algorithm.

5.1. Axiom Pinpointing in Pellet

Pellet exploits a tableau algorithm [68] that decides whether an axiom is entailed or not by a

KB $\mathcal{K}$ by refutation: axiom E is entailed if $\neg E$ has no model in the KB. The algorithm works on *completion graphs* also called *tableaux*: they are ABoxes that can also be seen as graphs, where each node represents an individual a and is labeled with the set of concepts $\mathcal{L}(a)$ it belongs to. Each edge $\langle a, b\rangle$ in the graph is labeled with the set of roles $\mathcal{L}(\langle a, b\rangle)$ to which the couple (a, b) belongs to. The algorithm starts from a tableau that contains the ABox of the KB and the negation of the axiom to be proved. For example, if the query is a membership one, $a : C$, it adds $\neg C$ to the label of a. If we query for the emptyness (unsatisfiability) of a concept C, the algorithm adds a new anonymous node a to the tableau and adds C to the label of a. The axiom $C \sqsubseteq D$ can be proved by showing that $C \sqcap \neg D$ is unsatisfiable. The algorithm repeatedly applies a set of consistency preserving *tableau expansion rules* until a clash (i.e., a contradiction) is detected or a clash-free graph is found to which no more rules are applicable.

In the following we describe the tableau algorithm used by Pellet for *concept unsatisfiability*, which checks if it is possible or not for a class to have any instances. Instance and subclass checking can be performed similarly. The algorithm is shown in Figure 5.

The rules used by Pellet to answer queries to $\mathcal{SHOIN}(\mathbf{D})$ knowledge bases are shown in Figure 8. Some of the rules are non-deterministic, i.e., they generate a finite set of tableaux. Thus the algorithm keeps a set of tableaux T. If a non-deterministic rule is applied to a graph G in T, then G is replaced by the resulting set of graphs. For example, if the disjunction $C \sqcup D$ is present in the label of a node, the rule $\rightarrow \sqcup$ generates two graphs, one in which C is added to the node label and the other in which D is added to the node label.

An *event* during the execution of the algorithm can be [28]: 1) $Add(C, a)$, the addition of a concept C to $\mathcal{L}(a)$; 2) $Add(R, \langle a, b\rangle)$, the addition of a role R to $\mathcal{L}(\langle a, b\rangle)$; 3) $Merge(a, b)$, the merging of the nodes a, b; 4) $\neq(a, b)$, the addition of the inequality $a \neq b$ to the relation $\neq$; 5) $Report(g)$, the detection of a clash g. We use $\mathcal{E}$ to denote the set of events recorded during the execution of the algorithm. A clash is either:

- a couple (C, a) where C and $\neg C$ are present in the label of node a, i.e. $\{C, \neg C\} \subseteq \mathcal{L}(a)$;
- a couple $(Merge(a, b), \neq(a, b))$, where the events $Merge(a, b)$ and $\neq(a, b)$ belong to $\mathcal{E}$.

Each time a clash is detected in a completion graph G, the algorithm stops applying rules to G. Once every completion graph in T contains a clash or no more expansion rules can be applied to it, the algorithm terminates. If all the completion graphs in the final set T contain a clash, the algorithm returns *unsatisfiable* as no model can be found. Otherwise, any one clash-free completion graph in T represents a possible model for $C(a)$ and the algorithm returns *satisfiable*.

Each expansion rule updates as well a *tracing function* τ, which associates sets of axioms with events in the derivation. For example, $\tau(Add(C, a))$, $(\tau(Add(R, \langle a, b\rangle)))$ is the set of axioms needed to explain the event $Add(C, a)$ $(Add(R, \langle a, b\rangle))$. For the sake of brevity, we define τ for couples (concept, individual) and (role, couple of individuals) as $\tau(C, a) = \tau(Add(C, a))$ and $\tau(R, \langle a, b\rangle) = \tau(Add(R, \langle a, b\rangle))$ respectively. The function τ is initialized as the empty set for all the elements of its domain except for $\tau(C, a)$ and $\tau(R, \langle a, b\rangle)$ to which the values $\{a : C\}$ and $\{(a, b) : R\}$ are assigned if $a : C$ and $(a, b) : R$ are in the ABox respectively. The expansion rules (Figure 8) add axioms to values of τ.

If $g_1, ..., g_n$ are the clashes, one for each tableau G of the final set, the output of the algorithm Tableau is $S = \bigcup_{i \in \{1,...,n\}} \tau(g_i) \setminus \{C(a)\}$ where a is the anonymous individual initially assigned to C. However, this set may be redundant because additional axioms may also be included in τ, e.g., during the $\rightarrow \leq$ rule, where axioms responsible for each of the successor edges are considered [28]. Thus S is pruned using a black-box approach shown in Figure 6 [28]. This algorithm executes a loop on S in which it removes one axiom from S in each iteration and checks whether the concept C turns satisfiable w.r.t. S, in which case the axiom is reinserted into S. The idea is that, if the concept turns satisfiable, we can conclude that the axiom extracted is responsible for the unsatisfiability and hence has to be inserted back into S. Vice-versa, if the concept still remains unsatisfiable, we can conclude that the axiom is irrelevant and can be removed from S. The process continues until all axioms in S have been tested and then returns S.

The algorithm for computing a single MinA, shown in Figure 7, first executes Tableau and then BlackBoxPruning.

```
1: function TABLEAU(C, K)
2:     Input: C (the concept to be tested for unsatisfiability)
3:     Input: K (the knowledge base)
4:     Output: S (a set of axioms) or null
5:     Let G_0 be an initial completion graph from K containing an anonymous individual a and C ∈ L(a)
6:     T ← {G_0}
7:     repeat
8:         Select a rule r applicable to a clash-free graph G from T
9:         T ← T \ {G}
10:        Let G = {G'_1, ..., G'_n} be the result of applying r to G
11:        T ← T ∪ G
12:    until all graphs in T have a clash or no rule is applicable
13:    if all graphs in T have a clash then
14:        S ← ∅
15:        for all G ∈ T do
16:            let s_G the result of τ for the clash of G
17:            S ← S ∪ s_G
18:        end for
19:        S ← S \ {C(a)}
20:        return S
21:    else
22:        return null
23:    end if
24: end function
```

Fig. 5. Tableau algorithm executed by Pellet.

```
1: function BLACKBOXPRUNING(C, S)
2:     Input: C (the concept to be tested for unsatisfiability)
3:     Input: S (the set of axioms to be pruned)
4:     Output: S (the pruned set of axioms)
5:     for all axiom E ∈ S do
6:         S ← S − {E}
7:         if C is satisfiable w.r.t. S then
8:             S ← S ∪ {E}
9:         end if
10:    end for
11:    return S
12: end function
```

Fig. 6. Black-Box pruning algorithm.

```
1: function SINGLEMINA(C, K)
2:     Input: C (the concept to be tested for unsatisfiability)
3:     Input: K (the knowledge base)
4:     Output: S (a MinA for the unsatisfiability of C w.r.t.
       K) or null
5:     S ←TABLEAU(C, K)
6:     if S = null then
7:         return null
8:     else
9:         return BLACKBOXPRUNING(C, S)
10:    end if
11: end function
```

Fig. 7. SingleMinA algorithm.

The output S of SINGLEMINA is guaranteed to be a MinA, as established by the following theorem, where ALL-MINAS$(C, \mathcal{K})$ stands for the set of MinAs in which C is unsatisfiable.

Theorem 4. *[28] Let C be an unsatisfiable concept w.r.t. $\mathcal{K}$ and let S be the output of the algorithm* SINGLEMINA *with input C and $\mathcal{K}$, then $S \in$* ALL-MINAS$(C, \mathcal{K})$.

SINGLEMINA returns a single MinA, i.e. a single explanation. To solve MIN-A-ENUM and find the remaining ones, Pellet exploits the *hitting set algorithm* [52].

Reiter considers a *universal set* U and a set $CS \subseteq \mathcal{P}U$ of *conflict sets*, where $\mathcal{P}$ denotes the powerset operator. The set $H \subseteq U$ is a *hitting set* for CS if each $c_i \in CS$ contains at least one element of H, i.e. if $c_i \cap H \neq \emptyset$ for all $1 \leq i \leq n$ (in other words, H 'hits' or intersects each set in CS). H is a *minimal hitting set* for CS if H is a hitting set for CS and no $H' \subset H$ is a hitting set for CS. The *hitting set problem* with input CS, U is to compute all the minimal hitting sets for CS. The idea here is that the explanations for unsatisfiability correspond to minimal conflict sets and an algorithm that generates minimal hitting sets can also be used to find all minimal conflict sets [28,30]. The universal set corresponds to the total set of axioms in the KB, and an explanation (for a particular concept unsatisfiability) corresponds to a single conflict set.

The algorithm constructs a labeled tree called *Hitting Set Tree* (HST) starting from a single explanation, the MinA S; then it removes each of the axioms in S individually, thereby creating new branches of the HST, and finds new explanations along these branches. The first step consists of initializing an HST $\mathbf{T} = (V, E, \mathcal{L})$ with S in the label of its root node, i.e. $V = \{v_0\}, E = \emptyset, \mathcal{L}(v_0) = S$. Then it selects an arbitrary axiom E in S, generates a new node w with an empty label in the tree and a new edge $\langle v_0, w \rangle$ with axiom E in its label. Then, the algorithm invokes SINGLEM-

$\rightarrow$ *unfold*: **if** $A \in \mathcal{L}(a)$, A atomic and $(A \sqsubseteq D) \in K$, **then**
if D $\notin \mathcal{L}(a)$, **then**
$Add(D, \mathcal{L}(a))$
$\tau(D, a) := (\tau(A, a) \cup \{A \sqsubseteq D\})$
$\rightarrow$ *CE*: **if** $(C \sqsubseteq D) \in K$, with C not atomic, a not blocked, **then**
if $(\neg C \sqcup D) \notin \mathcal{L}(a)$, **then**
$Add((\neg C \sqcup D), a)$
$\tau((\neg C \sqcup D), a) := \{C \sqsubseteq D\}$
$\rightarrow \sqcap$: **if** $(C_1 \sqcap C_2) \in \mathcal{L}(a)$, a is not indirectly blocked, **then**
if $\{C_1, C_2\} \not\subseteq \mathcal{L}(a)$, **then**
$Add(\{C_1, C_2\}, a)$
$\tau(C_i, a) := \tau((C_1 \sqcap C_2), a)$
$\rightarrow \sqcup$: **if** $(C_1 \sqcup C_2) \in \mathcal{L}(a)$, a is not indirectly blocked, **then**
if $\{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset$, **then**
Generate graphs $G_i := G$ for each $i \in \{1, 2\}$
$Add(C_i, a)$ in G_i for each $i \in \{1, 2\}$
$\tau(C_i, a) := \tau((C_1 \sqcup C_2), a)$
$\rightarrow \exists$: **if** $\exists S.C \in \mathcal{L}(a)$, a is not blocked, **then**
if a has no S-neighbor b with $C \in \mathcal{L}(b)$, **then**
create new node b, $Add(S, \langle a, b\rangle)$, $Add(C, b)$
$\tau(C, b) := \tau((\exists S.C), a)$
$\tau(S, \langle a, b\rangle) := \tau((\exists S.C), a)$
$\rightarrow \forall$: **if** $\forall(S.C) \in \mathcal{L}(a)$, a is not indirectly blocked and there is an S-neighbor b of a, **then**
if $C \notin \mathcal{L}(b)$, **then**
$Add(C, b)$
$\tau(C, b) := \tau((\forall S.C), a) \cup \tau(S, \langle a, b\rangle)$
$\rightarrow \forall^+$: **if** $\forall(S.C) \in \mathcal{L}(a)$, a is not indirectly blocked
and there is an R-neighbor b of a, $Trans(R)$ and $R \sqsubseteq S$, **then**
if $\forall R.C \notin \mathcal{L}(b)$, **then**
$Add(\forall R.C, b)$
$\tau((\forall R.C), b) := \tau((\forall S.C), a) \cup \tau(R, \langle a, b\rangle) \cup \{Trans(R)\} \cup \{R \sqsubseteq S\}$
$\rightarrow \geq$: **if** $(\geq nS) \in \mathcal{L}(a)$, a is not blocked, **then**
if there are no n safe S-neighbors $b_1, ..., b_n$ of a with $b_i \neq b_j$, **then**
create n new nodes $b_1, ..., b_n$; $Add(S, \langle a, b_i\rangle); \neq(b_i, b_j)$
$\tau(S, \langle a, b_i\rangle) := \tau((\geq nS), a)$
$\tau(\neq(b_i, b_j)) := \tau((\geq nS), a)$
$\rightarrow \leq$: **if** $(\leq nS) \in \mathcal{L}(a)$, a is not indirectly blocked,
and there are m S-neighbors $b_1, ..., b_m$ of a with $m > n$, **then**
For each possible pair b_i, b_j, $1 \leq i, j \leq m; i \neq j$ **then**
Generate a graph G'
$\tau(Merge(b_i, b_j)) := \tau((\leq nS), a) \cup \tau(S, \langle a, b_1\rangle) ... \cup \tau(S, \langle a, b_m\rangle)$
if b_j is a nominal node, **then** $Merge(b_i, b_j)$ in G',
else if b_i is a nominal node or ancestor of b_j, **then** $Merge(b_j, b_i)$
else $Merge(b_i, b_j)$ in G'
if b_i is merged into b_j, **then** for each concept C_i in $\mathcal{L}(b_i)$,
$\tau(C_i, b_j) := \tau(C_i, b_i) \cup \tau(Merge(b_i, b_j))$
(similarly for roles merged, and correspondingly for concepts in b_j if merged into b_i)
$\rightarrow O$: **if**, $\{o\} \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and not $a \neq b$, **then** $Merge(a, b)$
$\tau(Merge(a, b)) := \tau(\{o\}, a) \cup \tau(\{o\}, b)$
For each concept C_i in $\mathcal{L}(a)$, $\tau(Add(C_i, \mathcal{L}(b))) := \tau(Add(C_i, \mathcal{L}(a))) \cup \tau(Merge(a, b))$
(similarly for roles merged, and correspondingly for concepts in $\mathcal{L}(b)$)
$\rightarrow NN$: **if** $(\leq nS) \in \mathcal{L}(a)$, a nominal node, b blockable S-predecessor of a
and there is no m s.t. $1 \leq m \leq n$, $(\leq mS) \in \mathcal{L}(a)$
and there exist m nominal S-neighbors $c_1, ..., c_m$ of a s.t. $c_i \neq c_j$, $1 \leq j \leq m$, **then**
generate new G_m for each m, $1 \leq m \leq n$
and do the following in each G_m:
$Add(\leq mS, a)$, $\tau((\leq mS), a) := \tau((\leq nS), a) \cup (\tau(S, \langle b, a\rangle)$
create $b_1, ..., b_m$; add $b_i \neq b_j$ for $1 \leq i \leq j \leq m$.
$\tau(\neq(b_i, b_j) := \tau((\leq nS), a) \cup \tau(S, \langle b, a\rangle)$
$Add(S, \langle a, b_i\rangle)$; $Add(\{o_i\}, b_i)$;
$\tau(S, \langle a, b_i\rangle) := \tau((\leq nS), a) \cup \tau(S, \langle b, a\rangle)$; $\tau(\{o_i\}, b_i) := \tau((\leq nS), a) \cup \tau(S, \langle b, a\rangle)$

Fig. 8. Pellet tableau expansion rules for $\mathcal{SHOIN}(\mathbf{D})$ from [28].

INA with arguments C and the knowledge base $\mathcal{K}' = \mathcal{K} - \{E\}$, to test the unsatisfiability of C w.r.t. $\mathcal{K}'$. If C is still unsatisfiable it obtains a new explanation for C and it labels w with this set. The algorithm repeats this process (removing an axiom from S and executing SINGLEMINA to add a new node) until the unsatisfiability test returns negative (the concept turns satisfiable): in that case the algorithm labels the new node with OK and makes it a leaf. The algorithm also eliminates extraneous unsatisfiability tests based on previous results: once a path leading to a node labeled OK is found, any superset of that path is guaranteed to be a hitting set as well, and thus no additional unsatisfiability test is needed for that path, as indicated by a X in the node label. When the HST is fully built, all leaves of the tree are labeled with OK or X. The distinct non leaf nodes of the tree collectively represent the set ALL-MINAS$(C, \mathcal{K})$ of the unsatisfiable concept. The algorithm is shown in Figure 9. The correctness and completeness of the hitting set algorithm are given by the following theorem.

Theorem 5. *[28] Let C be a concept unsatisfiable in $\mathcal{K}$ and let* EXPHST$(C, \mathcal{K})$ *be the set of explanations returned by Pellet's hitting set algorithm. Then*

$$\text{EXPHST}(C, \mathcal{K}) = \text{ALL-MINAS}(C, \mathcal{K}).$$

Pellet's EXPHST$(C, \mathcal{K})$ can also take as input a maximum number of explanations to be generated. If the limit is reached during the execution of the hitting set algorithm, Pellet stops and returns the set of explanations found so far.

5.2. Overall BUNDLE

The main algorithm, shown in Figure 10, first builds a data structure $PMap$ that associates each probabilistic DL axiom E_i with its probability p_i. Then it uses Pellet's EXPHST$(C, \mathcal{K})$ function to compute the MinAs for the unsatisfiability of a concept C. These MinAs correspond to all conflict sets found by the Hitting Set Algorithm. BUNDLE exploits the version of this procedure in which we can specify the maximum number of explanations to be found.

Two data structures are initialized: $VarAx$ is an array that maintains the association between

```
1: procedure HITTINGSETTREE(C, K, CS, H, w, E, p)
2:     Input: C (the concept to be tested for unsatisfiability)
3:     Input: K (the knowledge base)
4:     Input/Output: CS (a set of Conflict Sets, initially con-
   taining a single explanation)
5:     Input/Output: H (a set of Hitting Sets)
6:     Input: w (the last node added to the Hitting Set Tree)
7:     Input: E (the last axiom removed from K)
8:     Input: p (the current edge path)
9:     if there exists a set h ∈ H s.t. (L(p) ∪ {E}) ⊆ h then
10:        L(w) ← X
11:        return
12:    else
13:        if C is unsatisfiable w.r.t. K then
14:            m ←SINGLEMINA(C, K)
15:            CS ← CS ∪ {m}
16:            create a new node w' and set L(w') ← m
17:            if w ≠ null then
18:                create an edge e = ⟨w, w'⟩ with L(e) = E
19:                p ← p ∪ e
20:            end if
21:            loop for each axiom F ∈ L(w')
22:                K' = K − {F}
23:                HITTINGSETTREE(C, K', CS, H, w', F, p)
24:            end loop
25:        else
26:            L(w) ← OK
27:            H ← H ∪ L(p)
28:        end if
29:    end if
30: end procedure
```

Fig. 9. Hitting Set Tree Algorithm.

Boolean random variables (whose index is the array index) and couples (axiom, probability), and BDD stores a BDD. BDD is initialized to the zero Boolean function (lines 8-9).

Then BUNDLE performs two nested loops that build a BDD representing the set of explanations. To manipulate BDDs we used JavaBDD[2] that is an interface to a number of underlying BDD manipulation packages. As the underlying package we used CUDD.

In the inner loop, BUNDLE generates the BDD for a single explanation, indicated as $BDDE$, which is initialized to the one Boolean function (lines 11-25). The axioms of each MinA are considered one by one. If the axiom is certain, then the one Boolean function is stored in $BDDA$ (line 14). Otherwise, the value p associated with the axiom Ax is extracted from $PMap$. The axiom is searched for in $VarAx$ to see if it has already been assigned a random variable. If not, a cell is added to $VarAx$ to store the couple (line 19). At this point we know the couple position i in the array $VarAx$, that is the index of its Boolean random variable X_i. We obtain a BDD representing $X_i = 1$ with BDDGETITHVAR in $BDDA$. $BDDA$ is fi-

[2] Available at `http://javabdd.sourceforge.net/`

```
1: function BUNDLE(𝒦, C, maxEx)
2:     Input: 𝒦 (the knowledge base)
3:     Input: C (the concept to be tested for unsatisfiability)
4:     Input: maxEx (the maximum number of explanations to find)
5:     Output: the probability of the unsatisfiability of C w.r.t. 𝒦
6:     Build Map PMap with sets of couples (axiom, probability)
7:     MinAs ←EXPHST(C, 𝒦, maxEx)          ▷ Pellet is called to compute the MinAs for the concept C
8:     Initialize VarAx to empty           ▷ VarAx is an array of couples (Axiom, Prob)
9:     BDD ←BDDZERO
10:    for all MinA ∈ MinAs do
11:        BDDE ←BDDONE
12:        for all Ax ∈ MinA do
13:            if 𝒦 contains a certain axiom Ax then
14:                BDDA ←BDDONE
15:            else
16:                p ← PMap(Ax)
17:                Scan VarAx looking for Ax
18:                if !found then
19:                    Add to VarAx a new cell containing (Ax, p)
20:                end if
21:                Let i be the position of (Ax, p) in VarAx
22:                BDDA ← BDDGETITHVAR(i)
23:            end if
24:            BDDE ←BDDAND(BDDE,BDDA)
25:        end for
26:        BDD ←BDDOR(BDD,BDDE)
27:    end for
28:    return PROB(BDD)                    ▷ VarAx is used to compute P(X) in PROB
29: end function
```

Fig. 10. Function BUNDLE: computation of the probability of an axiom C given the KB $\mathcal{K}$.

nally conjoined with the current $BDDE$ to get the BDD representing a single explanation (line 24).

In the outermost loop, BUNDLE combines BDDs for different explanations through disjunction between BDD and the current explanation $BDDE$ (line 26).

After the two cycles, function PROB of Figure 3 is called over BDD to return the probability of the query to the user.

We now prove BUNDLE correctness.

Theorem 6 (**BUNDLE correctness**). *Given a DISPONTE knowledge base $\mathcal{K}$, a query Q and a limit maxEx for the number of explanations to find, the probability returned by BUNDLE,* BUNDLE$(\mathcal{K}, Q, maxEx)$ *is:*

- *a lower bound on $P(Q)$ if a maximum number of explanations to compute is set, i.e.,* BUNDLE$(\mathcal{K}, Q, maxEx) \leq P(Q)$
- *equal to $P(Q)$, i.e.,* BUNDLE$(\mathcal{K}, Q, \infty) = P(Q)$ *otherwise*

Proof. Let K be EXPHST$(C, \mathcal{K}, maxEx)$. By Theorem 5

$$K \subseteq \text{ALL-MINAS}(C, \mathcal{K})$$

if a maximum number of explanations is set and

$$K = \text{ALL-MINAS}(C, \mathcal{K})$$

otherwise. Since BUNDLE computes $P(f_K(\mathbf{X}))$ for the Boolean function

$$f_K(\mathbf{X}) = \bigvee_{\kappa \in K} \bigwedge_{(F,1) \in \kappa} X_F$$

the theorem holds. □

Example 7. *Let us consider the KB presented in Example 5 and the query $C = kevin : NatureLover$. Let us also consider the corresponding $PMap = \{(fluffy : Cat, 0.4), (tom : Cat, 0.3), (Cat \sqsubseteq Pet, 0.6)\}$, $VarAx = \emptyset$ and the covering set of explanations $K = \{\kappa_1, \kappa_2\}$ where $\kappa_1 = \{(fluffy : Cat, 1), (Cat \sqsubseteq Pet, 1)\}$ and $\kappa_2 = \{(tom : Cat, 1), (Cat \sqsubseteq Pet, 1)\}$. Here we restrict explanations to contain only probabilistic axioms for the sake of simplicity.*

At the start, BUNDLE initializes BDD to the zero Boolean function and starts to loop over the explanations. It enters the inner loop considering the explanation κ_1 and initializing $BDDE$ to the one Boolean function. At this time $VarAx$ is empty, thus random variable X_1 is associated to $(fluffy : Cat, 0.4)$ by adding this pair to $VarAx$ in position 1. Function BDDGETITHVAR *returns a BDD in $BDDA$ that corresponds to the expression $X_1 = 1$. Then $BDDA$ is combined with $BDDE$ using the and operator. Now, the computation con-*

tinues by analyzing the second axiom of κ_1. The axiom $Cat \sqsubseteq Pet$ does not have a random variable associated with it yet, so the new variable X_2 is created and the pair $(Cat \sqsubseteq Pet, 0.6)$ is added to $VarAx$ in position 2. BDDA is then generated and combined with the BDDE corresponding to the current explanation using the and operator. Now all the axioms in κ_1 have been considered, so the final BDDE is combined with BDD with the or operator.

Then, BUNDLE starts to consider κ_2. BDDE is set to one. The axiom $Tom : Cat$ does not appear in $VarAx$ so variable X_3 is associated to $(Tom : Cat, 0.3)$ and BDDA, representing $X_3 = 1$, is joined with BDDE. The axiom $Cat \sqsubseteq Pet$ is found in $VarAx$ in position 2, so the function BDDGetIthVar *returns the BDD representing $X_2 = 1$. Finally BDDA is combined with the current BDDE. BDDE corresponding to the second explanation is completed so it is combined with BDD obtaining the one shown in Figure 2. Now BUNDLE calls function* Prob *which computes the probability and returns $P(C) = 0.348$.*

6. Computational Complexity

Jung and Lutz [27] considered the problem of computing the probability of conjunctive queries to probabilistic databases in the presence of an ontology. Probabilities can occur only in the ABox while the TBox is certain. In the case where each ABox assertion is associated with a Boolean random variable independent of all the others, they prove that only very simple conjunctive queries can be answered in PTime, while most queries are #P-hard when the ontology is a DL-Lite TBox and even when the ontology is an $\mathcal{ELI}$ TBox.

The class #P [72] describes counting problems associated with decision problems in NP. More formally, #P is the class of function problems of the form "compute $f(x)$", where f is the number of accepting paths of a nondeterministic Turing machine running in polynomial time. A prototypical #P problem is the one of computing the number of satisfying assignments of a CNF Boolean formula. #P problems were shown very hard. First, a #P problem must be at least as hard as the corresponding NP problem. Second, [70] showed that a polynomial-time machine with a #P oracle ($P^{\#P}$) can solve all problems in PH, the entire polynomial hierarchy.

The setting considered by [27] is subsumed by DISPONTE as it is equivalent to having probabilistic axioms only in the ABox of a DISPONTE KB. So the complexity result provides a lower bound for DISPONTE.

In order to investigate the complexity of BUNDLE, we can consider the two problems that it solves for answering a query. The first one is axiom pinpointing. Its computational complexity has been studied in a number of works [45,46,47]. Baader et al. [5] showed that there can be exponentially many MinAs for a very simple DL that allows only concept intersection.

Example 8. *Given an integer $n \geq 1$, consider the TBox*

$$\mathcal{T}_n = \{B_{i-1} \sqsubseteq P_i \sqcap Q_i, P_i \sqsubseteq B_i, Q_i \sqsubseteq B_i | 1 \leq i \leq n\}$$

The size of $\mathcal{T}_n$ is linear in n and $\mathcal{T}_n \models B_0 \sqsubseteq B_n$. There are 2^n MinAs for $B_0 \sqsubseteq B_n$ since, for each $i, 1 \leq i \leq n$, it is enough to have $P_i \sqsubseteq B_i$ or $Q_i \sqsubseteq B_i$ in the set.

Thus the number of explanations for $\mathcal{SHOIN}(\mathbf{D})$ may be even larger. Given this fact, we do not consider complexity with respect to the input only. We say an algorithm runs in *output polynomial time* [26] if it computes all the output in time polynomial in the overall size of the input and the output. Corollary 15 in [47] shows that MIN-A-ENUM cannot be solved in output polynomial time for *DL-Lite*$_{bool}$ TBoxes unless $P = NP$. Since *DL-Lite*$_{bool}$ is a sublogic of $\mathcal{SHOIN}(\mathbf{D})$, this result also holds for $\mathcal{SHOIN}(\mathbf{D})$.

The second problem to be solved is computing the probability of a query, that can be seen as computing the probability of a SUM-OF-PRODUCTS, as explained below.

Definition 2 (sum-of-products)**.** *Given a Boolean expression S in disjunctive normal form (DNF), or a sum-of-products, in the variables $\{V_1, \ldots, V_n\}$ and $P(V_i)$, the probability that V_i is true with $i = 1, \ldots, n$, compute the probability $P(S)$ of S, assuming all variables are independent.*

SUM-OF-PRODUCTS was shown to be #P-hard [see e.g. 51]. Given that the input of the SUM-OF-PRODUCTS problem is of at least exponential size in the worst case, this means that computing

the probability of an axiom from a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base is intractable.

However, the algorithms that have been proposed for solving the two problems were shown to be able to work on inputs of real world size. For example, all MinAs have been found for various entailments over many real world ontologies within a few seconds [28,30]. As regards the SUM-OF-PRODUCTS problem, algorithms based on BDDs were able to solve problems with hundreds of thousands of variables (see e.g. the works on inference on probabilistic logic programs [17,53,54,61, 32,63,62,55,56]). Also methods for weighted model counting [64,12] can be used to solve the SUM-OF-PRODUCTS problem.

Moreover, Section 8 shows that in practice we can compute the probability of entailments on KBs of real-world size with BUNDLE, too.

7. Related Work

Bacchus [6] and Halpern [23] discuss first-order logics of probability and distinguish statistical statements from statements about degrees of belief. Halpern [23] presents two examples: the probability that a randomly chosen bird flies is 0.9 and the probability that Tweety (a particular bird) flies is 0.9. The first statement captures statistical information about the world while the second captures a degree of belief. In order to express the second type of statement, called a "Type 2" statement, Halpern proposes the notation $w(Flies(tweety)) = 0.9$, where the function w is used to indicate the probability, while in order to express the first, called a "Type 1" statement, he proposes the notation $w_x(Flies(x) \wedge Bird(x)) = 0.9 w_x(Bird(x))$. The latter formula can be read as: given a randomly chosen x in the domain, if x is a bird, the probability that x flies is 0.9, or the conditional probability that x flies given that it is a bird is 0.9. DISPONTE allows to define only "Type 2" statements since the probability associated with an axiom represents the degree of belief in that axiom as a whole.

Lutz and Schröder [42] proposed Prob-$\mathcal{ALC}$ that is derived directly from Halpern's probabilistic first order logic and, as DISPONTE, considers only "Type 2" statements. They do so by adopting a possible world semantics and allowing concept expressions of the form $P_{\geq n}C$ and $\exists P_{\geq n}R.C$ in the language, the first expressing the set of individuals that belong to C with probability greater than n and the second the set of individuals a connected to at least another individual b of C by role R such that the probability of $R(a, b)$ is greater than n. Moreover, the ABox may contain expressions of the form $P_{\geq n}C(a)$ and $P_{\geq n}R(a, b)$ directly expressing degrees of belief, together with $P_{\geq n}\mathcal{A}$ where $\mathcal{A}$ is an ABox. Prob-$\mathcal{ALC}$ is 2-EXPTIME-hard even when probability values are restricted to 0 and 1. Prob-$\mathcal{ALC}$ is complementary to DISPONTE $\mathcal{ALC}$ as it allows new concept and assertion expressions while DISPONTE allows probabilistic axioms.

Jung and Lutz [27] presented an approach for querying probabilistic databases in the presence of an OWL2 QL ontology. Each assertion is assumed to be stored in a database and associated with probabilistic events. All atomic events are assumed to be probabilistically independent, resulting in a semantics very similar to the distribution semantics. The authors are interested in computing the answer probabilities to conjunctive queries. Probabilities can occur only in the data, but neither in the ontology nor in the query. Two types of ABoxes are considered: a general one where events are Boolean combinations of atomic events, and a restricted one, where each assertion is associated with a distinct atomic event. For Boolean conjunctive queries, the latter setting is subsumed by DISPONTE. Only very simple conjunctive queries in the latter setting can be answered in PTime, while most queries are #P-hard. The authors underline the general interest and usefulness of the approach for a wide range of applications including the management of data extracted from the web, machine translation, and dealing with data that arise from sensor networks.

Heinsohn [24] 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, Koller et al. [36] presented a probabilistic description logic based on Bayesian networks that deals with statistical terminological knowledge. Both Heinsohn and Koller et al. do not allow probabilistic assertional knowledge about concept and role instances. Jaeger [25] allows assertional knowledge about concept and role instances together with statistical terminological knowledge and combines the resulting probability distribu-

tions using cross-entropy minimization but does not allow "Type 2" statements as DISPONTE.

Ding and Peng [18] proposed a probabilistic extension of OWL that admits a translation into Bayesian networks. The semantics defines a probability distribution $P(a)$ over individuals, i.e. $\sum_a P(a) = 1$, and assigns a probability to a class C as $P(C) = \sum_{a \in C} P(a)$, while we assign a probability distribution to worlds. PR-OWL [13,11] is a probabilistic extension to OWL that consists of a set of classes, subclasses and properties that collectively form a framework for building probabilistic ontologies. It allows to use the first-order probabilistic logic MEBN [37] for representing uncertainty in ontologies.

DISPONTE differs from [24,25,36] because it provides a unified framework for representing different types of probabilistic knowledge: from assertional to terminological degree of belief knowledge.

DISPONTE differs from [18] because it specifies a distribution over worlds rather than individuals and from [13,11] because it does not resort to a full fledged first-order probabilistic language, allowing the reuse of inference technology from DLs.

Luna et al. [41] proposed an extension of $\mathcal{ALC}$, called CR$\mathcal{ALC}$, that adopts an interpretation-based semantics. CR$\mathcal{ALC}$ allows statistical axioms of the form $P(C|D) = \alpha$, which means that for any element x in the domain $\mathcal{D}$, the probability that x is in C given that it is in D is α, and of the form $P(R) = \beta$, which means that for each couple of elements x and y in $\mathcal{D}$, the probability that x is linked to y by the role R is β. The semantics of CR$\mathcal{ALC}$ is based on probability measures over the space of interpretations with a fixed domain. CR$\mathcal{ALC}$ allows to express "Type 1" knowledge but not "Type 2". A CR$\mathcal{ALC}$ KB $\mathcal{K}$ can be represented as a directed acyclic graph $G(\mathcal{K})$ in which a node represents a concept or a role and the edges represent the relations between them: if a concept C directly uses concept D, then D is a parent of C in $G(\mathcal{K})$. Moreover, each restriction $\exists R.C$ and $\forall R.C$ is added as a node to $G(\mathcal{K})$. $G(\mathcal{K})$ contains an edge from R to each restriction directly using it and from each restriction to the concept C appearing in it. $G(\mathcal{K})$ is a template for generating, given the domain $\mathcal{D}$, a ground graph in which each node represents an instantiated logical atom $C(a)$ or $R(a.b)$. Inference can then be performed by a first order loopy belief propagation algorithm on the ground graph.

A different approach to the combination of DLs with probability is taken in [19,39,40]. The logic proposed in these papers is called P-$\mathcal{SHIQ}(\mathbf{D})$ and allows both terminological probabilistic knowledge as well as assertional probabilistic knowledge about instances of concepts and roles. Terminological probabilistic knowledge is expressed using *conditional constraints* of the form $(D|C)[l, u]$ that informally mean "generally, if an object belongs to C, then it belongs to D with a probability in $[l, u]$". PRONTO [33,35] is a system that performs inference under this semantics. Similarly to [25], the terminological knowledge is interpreted statistically while the assertional knowledge is interpreted in an epistemic way by assigning degrees of beliefs to assertions. 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 [38]. These works are based on Nilsson's probabilistic logic [43], where a probabilistic interpretation Pr defines a probability distribution over the set of interpretations Int. The probability of a logical formula F according to Pr, denoted $Pr(F)$, is the sum of all $Pr(I)$ such that $I \in Int$ and $I \models F$. A probabilistic knowledge base $\mathcal{K}$ is a set of probabilistic formulas of the form $F \geq p$. A probabilistic interpretation Pr satisfies $F \geq p$ iff $Pr(F) \geq p$. Pr satisfies $\mathcal{K}$, or Pr is a model of $\mathcal{K}$, iff Pr satisfies all $F \geq p \in \mathcal{K}$. $Pr(F) \geq p$ is a tight logical consequence of $\mathcal{K}$ iff p is the infimum of $Pr(F)$ in the set of all models Pr of $\mathcal{K}$. 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 the first considers sets of distributions, the latter computes a single distribution over possible worlds. Nilsson's logic allows different conclusions: consider a DISPONTE KB composed of the axioms $0.4 :: a : C$ and $0.5 :: b : C$ and a probabilistic theory composed of $C(a) \geq 0.4$ and $C(b) \geq 0.5$; DISPONTE allows to say that $P(a : C \vee b : C) = 0.7$, while with Nilsson's logic the lowest p such that $Pr(C(a) \vee C(b)) \geq p$ holds is 0.5. This is due to the fact that, while in Nilsson's logic no assumption about the independence of the statements is made, in the distribution semantics the probabilistic axioms are considered as in-

dependent. While independencies can be encoded in Nilsson's logic by carefully choosing the values of the parameters, reading off the independencies from the theories becomes more difficult.

The assumption of independence of probabilistic axioms does not restrict expressiveness as one can specify any joint probability distribution over the logical ground atoms, possibly introducing new atoms if needed. This is testified by the fact that Bayesian networks can be encoded in probabilistic logic programs under the distribution semantics [75] and by the applications of the distribution semantics to a wide variety of domains [17,66,8].

For example, the Bayesian network in Figure 11 can be encoded with the following ProbLog program

```
burglary:0.1.
earthquake:0.2.
alarm :- burglary,earthquake.
alarm :- burglary,\+ earthquake,al_tf.
alarm :- \+ burglary,earthquake,al_ft.
alarm :- \+ burglary,\+ earthquake,al_ff.
al_tf:0.8.
al_ft:0.8.
al_ff:0.1.
```

where an additional probabilistic fact has been added for each row of the conditional probability table for `alarm` (excluding the first that models a deterministic dependency).

Other approaches, such as [14,20], combine a lightweight ontology language, *DL-Lite* and Datalog+/- respectively, with graphical models, Bayesian networks and Markov networks respectively. In both cases, a KB is composed of a set of annotated axioms and a graphical model. The annotations are sets of assignments of random variables from the graphical model. The semantics is assigned by considering the possible worlds of the graphical model and by stating that an axiom holds in a possible world if the assignments in its annotation hold. The probability of a conclusion is then the sum of the probabilities of the possible worlds where the conclusion holds. Our approach represents a special case of these in which each probabilistic axiom is annotated with a single random variable and the graphical model encodes independence among the random variables. This special case is of interest as inference technology from DLs can be directly employed.

8. Experiments

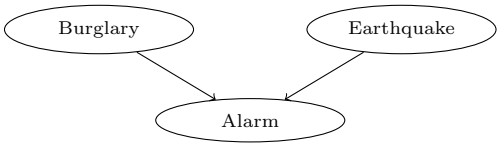
In order to test the performance of BUNDLE we performed several experiments.

8.1. Comparison with PRONTO

We compared BUNDLE with PRONTO by running queries w.r.t. increasingly complex ontologies. The experiments have been performed on Linux machines with a 3.10 GHz Intel Xeon E5-2687W with 2GB memory allotted to Java.

In the first experiment we generated the test ontologies by randomly sampling axioms from a large probabilistic ontology that models breast cancer risk assessment (BRCA) as shown in [34]. The main idea behind the design of the ontology was to reduce risk assessment to probabilistic entailment in P-$\mathcal{SHIQ}(\mathbf{D})$. The BRCA ontology contains a certain part and a probabilistic part. The tests were defined by randomly sampling axioms from the probabilistic part of this ontology which were then added to the certain part. So each sample is a probabilistic KB with the full certain part of the BRCA ontology and a subset of the probabilistic constraints. We varied the number of these constraints from 9 to 15, and, for each number, we generated 100 different consistent ontologies.

In order to generate a query, an individual a is added to the ontology: a is randomly assigned to each class that appears in the sampled conditional constraints with probability 0.6. If the class is composite, as for example *PostmenopausalWomanTakingTestosterone*, a is assigned to the component classes rather than to the composite one. In the example above, a will be added to *PostmenopausalWoman* and *WomanTakingTestosterone*. The ontologies were translated into DISPONTE by replacing each constraint $(D|C)[l, u]$ with the axiom $u :: C \sqsubseteq D$. For each ontology, we performed the query $a : C$ where the class C is randomly selected among those that represent women under increased and lifetime risk such as *WomanUnderLifetimeBRCRisk* and *WomanUnderStronglyIncreasedBRCRisk*. We then applied both BUNDLE and PRONTO to each generated test and we measured the execution time for inference and the memory used. Figure 12(a) shows the execution time averaged over the 100 KBs as a function of the number of probabilistic axioms and, similarly, Figure 12(b) shows



burg	t	f
	0.1	0.9

earthq	t	f
	0.2	0.8

alarm	t	f
b=t,e=t	1.0	0.0
b=t,e=f	0.8	0.2
b=f,e=t	0.8	0.2
b=f,e=f	0.1	0.9

Fig. 11. Bayesian network

the average amount of memory used. As one can see, inference times are similar for small knowledge bases, while the difference between the two reasoners rapidly increases for larger knowledge bases. The memory usage for BUNDLE is always less than 53% than PRONTO.

A second test was performed over larger KBs, following the method of [35]. We considered three different datasets: an extract from the Cell[3] ontology, an extract from the NCI Thesaurus[4] and an extract from the Teleost_anatomy[5] ontology. The Cell ontology represents cell types of the prokaryotic, fungal, and eukaryotic organisms. The NCI ontology is an extract from the NCI Thesaurus that describes human anatomy. The Teleost_anatomy (Teleost for short) ontology is a multi-species anatomy ontology for teleost fishes.

For each of these KBs, we considered the versions of increasing size used by [35]: they add 250, 500, 750 and 1,000 new probabilistic conditional constraints to the publicly available non-probabilistic version of each ontology. We converted these KBs into DISPONTE in the same way as for the BRCA ontology and we created a set of 100 different random subclass queries for each KB. For generating the queries we built the hierarchy of each KB and we randomly selected two classes connected in the hierarchy, so that each query had at least one explanation. We imposed a time limit of 5 minutes for BUNDLE to answer each query. If this limit is reached, BUNDLE's answer is "time-out".

In Table 1 we report, for each version of the datasets, the average execution time and the number of queries that terminated with a time-out (TO) for BUNDLE. The averages are computed on the queries that did not end with a time-out. In addition, for each KB we report its expressiveness and its number of non-probabilistic TBox axioms. In all these cases, PRONTO terminates with an out-of-memory error.

As can be seen, BUNDLE can manage larger KBs than PRONTO due to the lower amount of memory needed, as confirmed by the previous tests on BRCA. Moreover, BUNDLE answers most queries in a few seconds. However, some queries have a very high complexity that causes BUNDLE to reach the time-out, confirming the complexity results. In these cases, since the time-out is reached during the computation of the explanations, limiting the number of explanations is necessary, obtaining a lower bound on the probability that becomes tighter as more explanations are allowed.

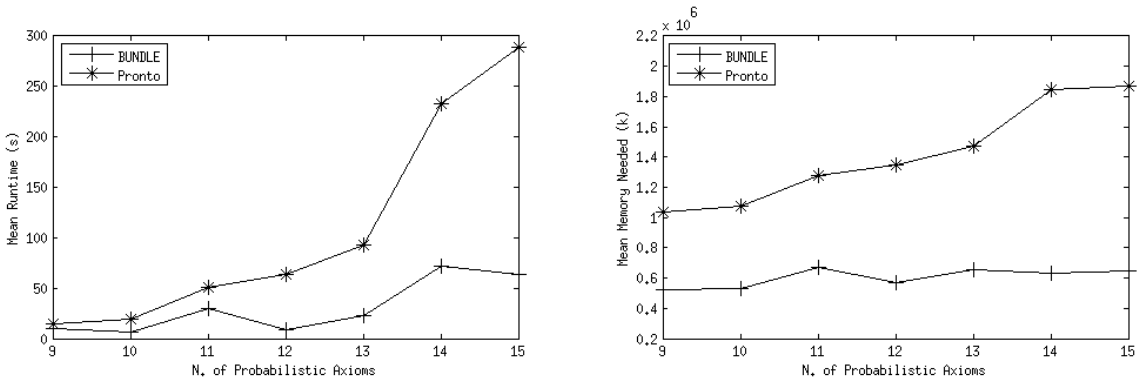
8.2. Tests with not entailed queries

In a further test we investigated cases for which subsumption does not hold. Thus, for the same versions of increasing size of the Cell, NCI and Teleost KBs we randomly created 100 different subclass queries that do not have explanations. In Table 2 we report for each KB the runtime in seconds. As for the previous test, we setted a time-out of 5 minutes but this limit was never reached.

8.3. Inference with a limit on the number of explanations

We studied how the execution time and the probability of queries vary when imposing a limit on the number of explanations. The experiments have been performed on Linux machines with a 3.10 GHz Intel Xeon E5-2687W with 2GB memory allotted to Java.

[3] http://cellontology.org/
[4] http://ncit.nci.nih.gov/
[5] http://phenoscape.org/wiki/Teleost_Anatomy_Ontology



(a) Average execution time (s) for inference.

(b) Average memory consumption (Kb) for inference.

Fig. 12. Comparison between BUNDLE and PRONTO on the BRCA KB.

Table 1
Average execution time for the queries to the Cell, Teleost and NCI KBs and number of queries terminated with a time-out (TO). The first column reports the expressiveness of each KB and the size of the non-probabilistic TBox.

		Size of the Probabilistic TBox				
Dataset & Infos		0	250	500	750	1,000
Cell	runtime (s)	0.76	2.84	3.88	3.94	4.53
$\mathcal{ALE}$+, 1,263 TBox axioms	TO	0	28	39	50	55
Teleost	runtime (s)	2.11	8.87	31.80	33.82	36.33
$\mathcal{ALEI}$+, 3,406 TBox axioms	TO	0	7	32	32	44
NCI	runtime (s)	3.02	11.37	11.37	16.37	24.90
$\mathcal{ALE}$+, 5,423 TBox axioms	TO	0	1	24	23	36

Table 2
Average execution time for the queries without explanations to the Cell, Teleost and NCI KBs.

		Size of the Probabilistic TBox				
Dataset		0	250	500	750	1,000
Cell	runtime (s)	0.47	1.52	2.43	2.52	3.14
Teleost	runtime (s)	1.15	4.51	12.76	14.69	15.27
NCI	runtime (s)	1.42	4.15	5.99	7.41	7.63

We chose the Grid[6] KB that is part of the myGrid project. The Grid KB has already been used for testing the performances of Pellet in [30]. It belongs to the bioinformatics domain and contains concepts at a high level of abstraction. For the test, we used a version of the Grid KB with $\mathcal{SHOIN}$ expressiveness that contains 2,838 axioms, 550 atomic concepts, 69 properties and 13 individuals, downloaded from the Tones repository[7]. We associated a probability of 0.5 to each axiom of the KB and then we ran 100 different subclass queries.

[6] http://www.myGrid.org.uk/

[7] http://rpc295.cs.man.ac.uk:8080/repository/browser

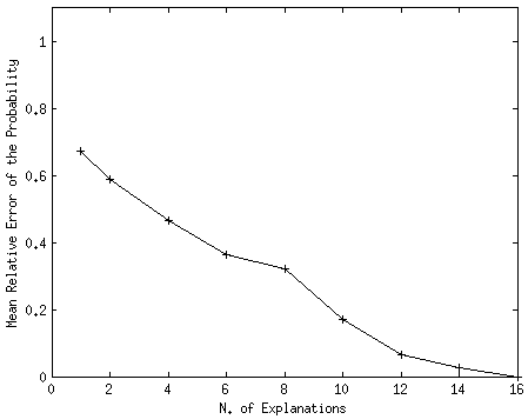


Fig. 13. Mean relative error of the probability of queries as a function of the limit on the number of explanations for the Grid KB.

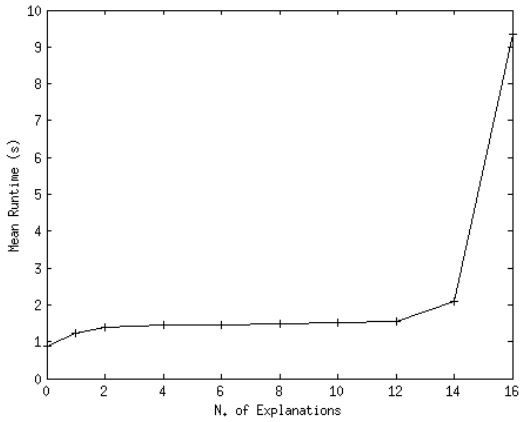


Fig. 14. Average execution time (s) as the limit on the number of explanations to the queries varies for the Grid KB.

We first computed the correct probability of each query by using BUNDLE without a limit on the number of explanations. Then we ran each query several times, each time with an increasing limit. The maximum number of explanations is 16: there were 20 queries with 16 explanations but most of the queries have a number of explanations between 1 and 5. We computed the relative error e between the correct probability p of a query and the probability p' returned by BUNDLE with a limit on the number of explanations with the formula $e = \frac{p-p'}{p}$. Then we averaged the relative error over all the queries.

In Figure 13 we show how the mean relative error varies with respect to the limit on the number of explanations. As can be seen, the quality of the answer increases as the limit on the number of explanations increases. Table 3 reports the execution times, averaged over all the 100 queries. The row of Table 3 with "–" in the first column contains the average execution time for BUNDLE without a limit on the number of explanations. Figure 14 shows the execution time as a function of the limit on the number of explanations, based on the values of Table 3.

8.4. Scalability of BUNDLE

Finally, we did a scalability test with two different KBs: the full version of the NCI Thesaurus (NCI_full for short) with $\mathcal{SH}$ expressiveness that contains 3,382,017 axioms and the Foundational Model of Anatomy Ontology (FMA for short)[8] with $\mathcal{ALCOIN}(\mathbf{D})$ expressiveness. FMA is a KB for biomedical informatics that models the phenotypic structure of the human body anatomy. It contains 88,252 axioms in the TBox and RBox and 237,382 individuals. The experiments have been performed on a Linux machine with a 2.33 GHz Intel Dual Core E6550 with 2GB memory allotted to Java.

For NCI_full we generated 10 ontologies of increasing size that contain $10\%, ..., 100\%$ of the axioms. Then we randomly selected an increasing number of certain axioms from these subontologies and made them probabilistic. We sampled 5,000, 10,000, 15,000, 20,000, 25,000 different probabilistic axioms, obtaining 50 different probabilistic KBs with total size from 338,201 to 3,382,017 axioms. Then we randomly created 100 subclass queries for each of the 50 subontologies and ran them. Figure 15 shows the trend of the runtime averaged over the queries with respect to the total size of the ontologies and the subset of probabilistic axioms. The maximum time spent for computing the probability of a query is 266.24 seconds with the KB that contains 90% of the axioms.

Finally, we exploited the FMA ontology for running a scalability test where only the size of the ABox varies. We generated versions of the ontol-

[8] `http://sig.biostr.washington.edu/projects/fm/index.html`

Table 3
Average execution time depending on the limit on the number of explanations for the Grid KB. The last row reports the time spent for finding the set of all explanations.

Limit on the explanations	Runtime (s)
0	0.81
2	1.40
4	1.44
6	1.46
8	1.49
10	1.52
12	1.55
14	2.10
16	9.36
–	18.44

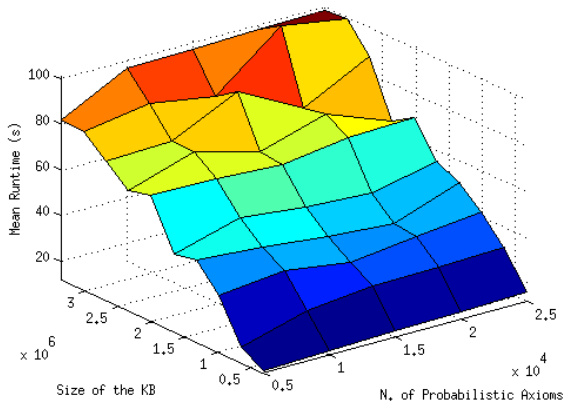


Fig. 15. Average execution time (s) for the queries to the NCI_full KB on versions of increasing size of the ontology and of the probabilistic part. On the x axis is reported the total number of axioms of the KB (including the probabilistic ones) and on the y axis the number of probabilistic axioms considered.

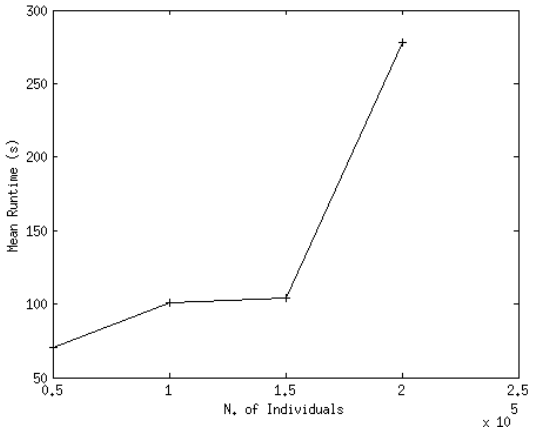


Fig. 16. Average execution time (s) for the queries to the FMA KB with respect to the increasing size of the ABox. BUNDLE cannot manage the entire ABox (237382 individuals).

ogy that contain the entire TBox and RBox, 500 probabilistic axioms and an increasing number of individuals. The size of the ABox varies between 50,000 and all the individuals contained in the full ABox with a step of 50,000. We generated 100 instance-of queries by randomly selecting an individual and a class among those to which it belongs. Figure 16 shows how the runtime averaged over the queries varies with respect to the size of the ABox. With 237382 individuals, that correspond to the entire ABox, BUNDLE raises an out-of-memory error. The maximum inference time reached is 298.98 seconds with 200,000 individuals.

As can be seen, BUNDLE can manage very large ontologies and answer queries in a relatively short time.

9. Conclusions

We have presented the DISPONTE semantics for probabilistic DLs that is inspired by the distribution semantics of probabilistic logic programming. We have discussed the application of DISPONTE to $\mathcal{SHOIN}(\mathbf{D})$, a prototype of an expressive DL. DISPONTE minimally extends the language to which it is applied and allows both assertional and terminological probabilistic

knowledge. We have also presented the algorithm BUNDLE that is able to compute the probability of queries from DISPONTE KBs. BUNDLE has been both compared with the probabilistical reasoner PRONTO and tested for scalability on several real world KBs. The experiments show that BUNDLE is able to deal with ontologies of significant complexity, even if in some situations only an approximated answer takes a reasonable amount of time. BUNDLE is available for download from `http://sites.unife.it/ml/bundle` together with the datasets used in the experiments.

In the future, we plan to consider extensions of the semantics including statistical or "Type 1" knowledge in the terminology of [23]. Moreover, alternative approaches for inference will be considered, in particular reasoning algorithms returning a *pinpointing formula* [4,3]: such a formula compactly encodes explanations for the query and can be converted directly into a BDD. We will also consider non-Boolean conjunctive queries and develop inference algorithms for answering them. One possibility would be to first retrieve the instances satisfying the query and then compute explanations for each of them. We have also started to study the problem of learning the parameters [60] and we are planning to tackle the problem of learning the structure of probabilistic KBs.

References

[1] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F. (Eds.), 2003. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.

[2] Baader, F., Horrocks, I., Sattler, U., 2008. Description logics. In: Handbook of knowledge representation. Elsevier, Ch. 3, pp. 135–179.

[3] Baader, F., Peñaloza, R., 2010. Automata-based axiom pinpointing. Journal of Automated Reasoning 45 (2), 91–129.

[4] Baader, F., Peñaloza, R., 2010. Axiom pinpointing in general tableaux. Journal of Logic and Computation 20 (1), 5–34.

[5] Baader, F., Peñaloza, R., Suntisrivaraporn, B., 2007. Pinpointing in the description logic $\mathcal{EL}^+$. In: Annual German Conference on AI. Vol. 4667 of LNCS. Springer, pp. 52–67.

[6] Bacchus, F., 1990. Representing and reasoning with probabilistic knowledge - a logical approach to probabilities. MIT Press.

[7] Bellodi, E., Lamma, E., Riguzzi, F., Albani, S., 2011. A distribution semantics for probabilistic ontologies. In: International Workshop on Uncertain Reasoning for the Semantic Web. Vol. 778 of CEUR Workshop Proceedings. Sun SITE Central Europe, pp. 75–86.

[8] Bellodi, E., Riguzzi, F., 2012. Experimentation of an expectation maximization algorithm for probabilistic logic programs. Intelligenza Artificiale 8 (1), 3–18.

[9] Bellodi, E., Riguzzi, F., 2013. Expectation Maximization over binary decision diagrams for probabilistic logic programs. Intelligent Data Analysis 17 (2), 343–363.

[10] Bollig, B., Wegener, I., 1996. Improving the variable ordering of OBDDs is NP-complete. IEEE Transactions on Computers 45 (9), 993–1002.

[11] Carvalho, R. N., Laskey, K. B., Costa, P. C. G., 2010. PR-OWL 2.0 - bridging the gap to OWL semantics. In: International Workshop on Uncertain Reasoning for the Semantic Web. Vol. 654 of CEUR Workshop Proceedings. Sun SITE Central Europe, pp. 73–84.

[12] Chavira, M., Darwiche, A., 2008. On probabilistic inference by weighted model counting. Artificial Intelligence 172 (6-7), 772–799.

[13] Costa, P. C. G., Laskey, K. B., Laskey, K. J., 2008. PR-OWL: A Bayesian ontology language for the semantic web. In: International Workshop on Uncertainty Reasoning for the Semantic Web. Vol. 5327 of LNCS. Springer, pp. 88–107.

[14] d'Amato, C., Fanizzi, N., Lukasiewicz, T., 2008. Tractable reasoning with Bayesian description logics. In: International Conference on Scalable Uncertainty Management. Vol. 5291 of LNCS. Springer, pp. 146–159.

[15] Darwiche, A., Marquis, P., 2002. A knowledge compilation map. Journal of Artificial Intelligence Research 17, 229–264.

[16] De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J., 2008. Towards digesting the alphabet-soup of statistical relational learning. In: Workshop on Probabilistic Programming. pp. 1–14.

[17] De Raedt, L., Kimmig, A., Toivonen, H., 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence. pp. 2462–2467.

[18] Ding, Z., Peng, Y., 2004. A probabilistic extension to ontology language OWL. In: Hawaii International Conference On System Sciences. IEEE, pp. 1–10.

[19] Giugno, R., Lukasiewicz, T., 2002. P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web. In: European Conference on Logics in Artificial Intelligence. Vol. 2424 of LNCS. Springer, pp. 86–97.

[20] Gottlob, G., Lukasiewicz, T., Simari, G. I., 2011. Conjunctive query answering in probabilistic Datalog+/- ontologies. In: International Conference on Web Reasoning and Rule Systems. Vol. 6902 of LNCS. Springer, pp. 77–92.

[21] Grumberg, O., Livne, S., Markovitch, S., 2003. Learning to order bdd variables in verification. Journal of

Artificial Intelligence Research 18, 83–116.
[22] Halaschek-Wiener, C., Kalyanpur, A., Parsia, B., 2006. Extending tableau tracing for ABox updates. Tech. rep., University of Maryland.
[23] Halpern, J. Y., 1990. An analysis of first-order logics of probability. Artificial Intelligence 46 (3), 311–350.
[24] Heinsohn, J., 1994. Probabilistic description logics. In: Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann, pp. 311–318.
[25] Jaeger, M., 1994. Probabilistic reasoning in terminological logics. In: International Conference on Principles of Knowledge Representation and Reasoning. pp. 305–316.
[26] Johnson, D. S., Papadimitriou, C. H., Yannakakis, M., 1988. On generating all maximal independent sets. Information Processing Letters 27 (3), 119–123.
[27] Jung, J. C., Lutz, C., 2012. Ontology-based access to probabilistic data with owl ql. In: International Semantic Web Conference. Vol. 7649 of LNCS. Springer, pp. 182–197.
[28] Kalyanpur, A., 2006. Debugging and repair of OWL ontologies. Ph.D. thesis, The Graduate School of the University of Maryland.
[29] Kalyanpur, A., Parsia, B., Cuenca-Grau, B., Sirin, E., 2005. Tableaux tracing in SHOIN. Tech. Rep. 2005-66, University of Maryland.
[30] Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E., 2007. Finding all justifications of OWL DL entailments. In: International Semantic Web Conference. Vol. 4825 of LNCS. Springer, pp. 267–280.
[31] Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J. A., 2005. Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics 3 (4), 268–293.
[32] Kimmig, A., Demoen, B., De Raedt, L., Costa, V. S., Rocha, R., 2011. On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming 11 (2-3), 235–262.
[33] Klinov, P., 2008. Pronto: A non-monotonic probabilistic description logic reasoner. In: European Semantic Web Conference. Vol. 5021 of LNCS. Springer, pp. 822–826.
[34] Klinov, P., Parsia, B., 2008. Optimization and evaluation of reasoning in probabilistic description logic: Towards a systematic approach. In: International Semantic Web Conference. Vol. 5318 of LNCS. Springer, pp. 213–228.
[35] Klinov, P., Parsia, B., 2011. A hybrid method for probabilistic satisfiability. In: International Conference on Automated Deduction. Vol. 6803 of LNCS. Springer, pp. 354–368.
[36] Koller, D., Levy, A. Y., Pfeffer, A., 1997. P-CLASSIC: A tractable probablistic description logic. In: National Conference on Artificial Intelligence. pp. 390–397.
[37] Laskey, K. B., Costa, P. C. G., 2005. Of starships and Klingons: Bayesian logic for the 23rd century. In: Conference in Uncertainty in Artificial Intelligence. AUAI Press, pp. 346–353.
[38] Lehmann, D. J., 1995. Another perspective on default reasoning. Annals of Mathematics and Artificial Intelligence 15 (1), 61–82.
[39] Lukasiewicz, T., 2002. Probabilistic default reasoning with conditional constraints. Annals of Mathematics and Artificial Intelligence 34 (1-3), 35–88.
[40] Lukasiewicz, T., 2008. Expressive probabilistic description logics. Artificial Intelligence 172 (6-7), 852–883.
[41] Luna, J. E. O., Revoredo, K., Cozman, F. G., 2011. Learning probabilistic description logics: A framework and algorithms. In: Mexican International Conference on Artificial Intelligence. Vol. 7094 of LNCS. Springer, pp. 28–39.
[42] Lutz, C., Schröder, L., 2010. Probabilistic description logics for subjective uncertainty. In: International Conference on Principles of Knowledge Representation and Reasoning. AAAI Press, pp. 393–403.
[43] Nilsson, N. J., 1986. Probabilistic logic. Artificial Intelligence 28 (1), 71–87.
[44] Patel-Schneider, P, F., Horrocks, I., Bechhofer, S., 2003. Tutorial on OWL.
[45] Peñaloza, R., Sertkaya, B., 2009. Axiom pinpointing is hard. In: International Workshop on Description Logics. Vol. 477 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 1–12.
[46] Peñaloza, R., Sertkaya, B., 2010. Complexity of axiom pinpointing in the DL-Lite family. In: International Workshop on Description Logics. Vol. 573 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 1–12.
[47] Peñaloza, R., Sertkaya, B., 2010. Complexity of axiom pinpointing in the DL-Lite family of description logics. In: European Conference on Artificial Intelligence. IOS Press, pp. 29–34.
[48] Poole, D., 1993. Probabilistic horn abduction and Bayesian networks. Artificial Intelligence 64 (1), 81–129.
[49] Poole, D., 1997. The Independent Choice Logic for modelling multiple agents under uncertainty. Artificial Intelligence 94 (1-2), 7–56.
[50] Poole, D., 2000. Abducing through negation as failure: stable models within the independent choice logic. Journal of Logic Programming 44 (1-3), 5–35.
[51] Rauzy, A., Châtelet, E., Dutuit, Y., Bérenguer, C., January 2003. A practical comparison of methods to assess sum-of-products. Reliability Engineering and System Safety 79 (1), 33–42.
[52] Reiter, R., 1987. A theory of diagnosis from first principles. Artificial Intelligence 32 (1), 57–95.
[53] Riguzzi, F., 2007. A top down interpreter for LPAD and CP-logic. In: Congress of the Italian Association for Artificial Intelligence. Vol. 4733 of LNAI. Springer, pp. 109–120.
[54] Riguzzi, F., 2009. Extended semantics and inference for the Independent Choice Logic. Logic Journal Of The IGPL 17 (6), 589–629.
[55] Riguzzi, F., 2013. MCINTYRE: A Monte Carlo system for probabilistic logic programming. Fundamenta Informaticae 124 (4), 521–541.
[56] Riguzzi, F., 2014. Speeding up inference for probabilistic logic programs. The Computer Journal 57 (3), 347–363.
[57] Riguzzi, F., Bellodi, E., Lamma, E., 2012. Probabilistic Datalog+/- under the distribution semantics. In: International Workshop on Description Logics. Vol. 846

of CEUR Workshop Proceedings. Sun SITE Central Europe, pp. 1–11.

[58] Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., 2012. Epistemic and statistical probabilistic ontologies. In: International Workshop on Uncertain Reasoning for the Semantic Web. Vol. 900 of CEUR Workshop Proceedings. Sun SITE Central Europe, pp. 3–14.

[59] Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., 2013. BUNDLE: A reasoner for probabilistic ontologies. In: International Conference on Web Reasoning and Rule Systems. Vol. 7994 of LNCS. Springer, pp. 183–197.

[60] Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., 2013. Parameter learning for probabilistic ontologies. In: International Conference on Web Reasoning and Rule Systems. Vol. 7994 of LNCS. Springer, pp. 265–270.

[61] Riguzzi, F., Swift, T., 2010. Tabling and Answer Subsumption for Reasoning on Logic Programs with Annotated Disjunctions. In: Technical Communications of the International Conference on Logic Programming. Vol. 7 of Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 162–171.

[62] Riguzzi, F., Swift, T., 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. Theory and Practice of Logic Programming 11 (Special Issue 4–5 - 27th International Conference on Logic Programming), 433–449.

[63] Riguzzi, F., Swift, T., 2013. Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. Theory and Practice of Logic Programming 13 (Special Issue 02 - 25th Annual GULP Conference), 279–302.

[64] Sang, T., Beame, P., Kautz, H. A., 2005. Performing bayesian inference by weighted model counting. In: National Conference on Artificial Intelligence. AAAI Press / The MIT Press, pp. 475–482.

[65] Sato, T., 1995. A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. MIT Press, pp. 715–729.

[66] Sato, T., Kameya, Y., 2001. Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research 15, 391–454.

[67] Schlobach, S., Cornet, R., 2003. Non-standard reasoning services for the debugging of description logic terminologies. In: International Joint Conference on Artificial Intelligence. Morgan Kaufmann, pp. 355–362.

[68] Schmidt-Schauß, M., Smolka, G., 1991. Attributive concept descriptions with complements. Artificial Intelligence 48 (1), 1–26.

[69] Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y., 2007. Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5 (2), 51–53.

[70] Toda, S., 1989. On the computational power of PP and +P. In: Annual Symposium on Foundations of Computer Science. IEEE Computer Society, pp. 514–519.

[71] URW3-XG, 2008. Uncertainty reasoning for the World Wide Web, final report.

[72] Valiant, L. G., 1979. The complexity of enumeration and reliability problems. SIAM Journal on Computing 8 (3), 410–421.

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

[74] Vennekens, J., Verbaeten, S., 2003. Logic programs with annotated disjunctions. Tech. Rep. CW386, KU Leuven.

[75] Vennekens, J., Verbaeten, S., Bruynooghe, M., 2004. Logic programs with annotated disjunctions. In: International Conference on Logic Programming. Vol. 3131 of LNCS. Springer, pp. 195–209.