A Modular Inference System for Probabilistic Description Logics

Giuseppe Cota[1(✉)], Fabrizio Riguzzi[2], Riccardo Zese[1], Elena Bellodi[2], and Evelina Lamma[1]

[1] Dipartimento di Ingegneria, University of Ferrara, Via Saragat 1, 44122 Ferrara, Italy
{giuseppe.cota,fabrizio.riguzzi,riccardo.zese,elena.bellodi, evelina.lamma}@unife.it
[2] Dipartimento di Matematica e Informatica, University of Ferrara, Via Saragat 1, 44122 Ferrara, Italy

Abstract. While many systems exist for reasoning with Description Logics knowledge bases, very few of them are able to cope with uncertainty. BUNDLE is a reasoning system, exploiting an underlying non-probabilistic reasoner (Pellet), able to perform inference w.r.t. Probabilistic Description Logics. In this paper, we report on a new *modular* version of BUNDLE that can use other OWL (non-probabilistic) reasoners and various approaches to perform probabilistic inference. BUNDLE can now be used as a standalone desktop application or as a library in OWL API-based applications that need to reason over Probabilistic Description Logics. Due to the introduced modularity, BUNDLE performance now strongly depends on the method and OWL reasoner chosen to obtain the set of justifications. We provide an evaluation on several datasets as the inference settings vary.

Keywords: Probabilistic Description Logic · Semantic Web Reasoner · OWL Library

1 Introduction

The aim of the Semantic Web is to make information available in a form that is understandable and automatically manageable by machines. In order to realize this vision, the W3C has supported the development of a family of knowledge representation formalisms of increasing complexity for defining ontologies, called OWL (Web Ontology Language), that are based on Description Logics (DLs). Many inference systems, generally called reasoners, have been proposed to reason upon these ontologies, such as Pellet [23], Hermit [22] and Fact++ [24].

Nonetheless, modeling real-world domains requires dealing with information that is incomplete or that comes from sources with different trust levels. This motivates the need for the uncertainty management in the Semantic Web, and many proposals have appeared for combining probability theory with OWL languages, or with the underlying DLs [4,8,12,14,15]. Among them, in [18,26] we

D. Ciucci et al. (Eds.): SUM 2018, LNAI 11142, pp. 78–92, 2018.
https://doi.org/10.1007/978-3-030-00461-3_6

introduced the DISPONTE semantics for probabilistic DLs. DISPONTE borrows the distribution semantics [20] from Probabilistic Logic Programming, that has emerged as one of the most effective approaches for representing probabilistic information in Logic Programming languages. Examples of probabilistic reasoners that perform inference under DISPONTE are BUNDLE [18,19,26], TRILL and TRILL^{P} [26,27]. The first one is implemented in Java, whereas the other two are written in Prolog to exploit Prolog's backtracking facilities during the search of all the possible justifications.

In order to perform probabilistic inference over DISPONTE knowledge bases (KBs), it is necessary to find the covering set of justifications and this is accomplished by a non probabilistic reasoner. The first version of BUNDLE was able to execute this search by exploiting only the Pellet reasoner [23].

In this paper, we propose a new version of BUNDLE which is modular and allows one to use different OWL reasoners and different approaches for justification finding. In particular, it embeds Pellet, Hermit, Fact++ and JFact as OWL reasoners, and three justification generators, namely GlassBox (only for Pellet), BlackBox and OWL Explanation. The introduced modularity has two main advantages with respect to BUNDLE's previous version. First, it allows one to "plug-in" a new OWL API-based reasoner in a very simple manner. Second, the framework can be easily extended by including new concrete implementations of algorithms for justification finding.

In this modular version, BUNDLE performance will strongly depend on the sub-system employed to build the set of justifications for a given query. To evaluate it we ran several experiments on different real-world and synthetic datasets.

The paper is organized as follows: Sect. 2 briefly introduces DLs, while Sect. 3 illustrates the justification finding problem. Sections 4 and 5 present DISPONTE and the theoretical aspects of inference in DISPONTE KBs respectively. The description of BUNDLE is provided in Sect. 6. Finally, Sect. 7 shows the experimental evaluation and Sect. 8 concludes the paper.

2 Description Logics

An ontology describes the concepts of the domain of interest and their relations with a formalism that allows information to be processable by machines. The *Web Ontology Language* (OWL) is a family of knowledge representation languages for authoring ontologies or knowledge bases. OWL 2 [25] is the last version of this language and since 2012 it became a W3C recommendation.

Descriptions Logics (DLs) provide a logical formalism for knowledge representation. They are useful in all the domains where it is necessary to represent information and to perform inference on it, such as software engineering, medical diagnosis, digital libraries, databases and Web-based informative systems. They possess nice computational properties such as decidability and (for some DLs) low complexity [1].

There are many different DL languages that differ in the constructs that are allowed for defining concepts (sets of individuals of the domain) and roles

(sets of pairs of individuals). The $\mathcal{SROIQ}(\mathbf{D})$ DL is one of the most common fragments; it was introduced by Horrocks et al. in [7] and it is of particular importance because it is semantically equivalent to OWL 2.

Let us consider a set of *atomic concepts* $\mathbf{C}$, a set of *atomic roles* $\mathbf{R}$ and a set of individuals $\mathbf{I}$. A *role* could be an atomic role $R \in \mathbf{R}$, the inverse R^- of an atomic role $R \in \mathbf{R}$ or a complex role $R \circ S$. We use $\mathbf{R}^-$ to denote the set of all inverses of roles in $\mathbf{R}$. Each $A \in \mathbf{A}$, $\bot$ and $\top$ are concepts and if $a \in \mathbf{I}$, then $\{a\}$ is a concept called *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 $\exists R.C$, $\forall R.C$, $\geq nR.C$ and $\leq nR.C$ for an integer $n \geq 0$.

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

A KB is usually assigned a semantics using interpretations of the form $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, 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 individual a, a subset of $\Delta^\mathcal{I}$ to each concept C and a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ to each role R. The mapping $\cdot^\mathcal{I}$ is extended to complex concepts as follows (where $R^\mathcal{I}(x, C) = \{y | \langle x, y \rangle \in R^\mathcal{I}, y \in C^\mathcal{I}\}$ and $\#X$ denotes the cardinality of the set X):

$$\top^\mathcal{I} = \Delta^\mathcal{I} \qquad \bot^\mathcal{I} = \emptyset$$
$$\{a\}^\mathcal{I} = \{a^\mathcal{I}\} \qquad (\neg C)^\mathcal{I} = \Delta^\mathcal{I} \backslash C^\mathcal{I}$$
$$(C_1 \sqcup C_2)^\mathcal{I} = C_1^\mathcal{I} \cup C_2^\mathcal{I} \qquad (C_1 \sqcap C_2)^\mathcal{I} = C_1^\mathcal{I} \cap C_2^\mathcal{I}$$
$$(\exists R.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} | R^\mathcal{I}(x) \cap C^\mathcal{I} \neq \emptyset\} \qquad (\forall R.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} | R^\mathcal{I}(x) \subseteq C^\mathcal{I}\}$$
$$(\geq nR.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} | \#R^\mathcal{I}(x, C) \geq n\} \qquad (\leq nR.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} | \#R^\mathcal{I}(x, C) \leq n\}$$
$$(R^-)^\mathcal{I} = \{(y, x) | (x, y) \in R^\mathcal{I}\} \qquad (R_1 \circ \ldots \circ R_n)^\mathcal{I} = R_1^\mathcal{I} \circ \ldots \circ R_n^\mathcal{I}$$

$\mathcal{SROIQ}(\mathbf{D})$ also permits the definition of datatype roles, which connect an individual to an element of a datatype such as integers, floats, etc.

A query Q over a KB $\mathcal{K}$ is usually an axiom for which we want to test the entailment from the KB, written as $\mathcal{K} \models Q$.

Example 1. Consider the following KB "Crime and Punishment"

$$\mathsf{Nihilist} \sqsubseteq \mathsf{GreatMan} \qquad \exists \mathsf{killed}.\top \sqsubseteq \mathsf{Nihilist}$$
$$(\mathsf{raskolnikov}, \mathsf{alyona}) : \mathsf{killed} \qquad (\mathsf{raskolnikov}, \mathsf{lizaveta}) : \mathsf{killed}$$

This KB states that if you killed someone then you are a nihilist and whoever is a nihilist is a "great man" (TBox). It also states that Raskolnikov killed Alyona and Lizaveta (ABox). The KB entails the query $Q = \mathsf{raskolnikov} : \mathsf{GreatMan}$ (but are we sure about that?).

3 Justification Finding Problem

Here we discuss the problem of finding the covering set of justifications for a given query. This non-standard reasoning service is also known as *axiom pinpointing* [21] and it is useful for tracing derivations and debugging ontologies. This problem has been investigated by various authors [2,6,9,21]. A justification corresponds to an *explanation* for a query Q. An explanation is a subset of logical axioms $\mathcal{E}$ of a KB $\mathcal{K}$ such that $\mathcal{E} \models Q$, whereas a justification is an explanation such that it is minimal w.r.t. set inclusion. Formally, we say that an explanation $\mathcal{J} \subseteq \mathcal{K}$ is a justification if for all $\mathcal{J}' \subset \mathcal{J}$, $\mathcal{J}' \not\models Q$, i.e. $\mathcal{J}'$ is not an explanation for Q. The problem of enumerating all justifications that entail a given query is called axiom pinpointing or justification finding. *The set of all the justifications for the query Q is the covering set of justifications for Q*. Given a KB $\mathcal{K}$, the covering set of justifications for Q is denoted by $\textsc{All-Just}(Q, \mathcal{K})$.

Below, we provide the formal definitions of justification finding problem.

Definition 1 (Justification finding problem).
***Input**: A knowledge base $\mathcal{K}$, and an axiom Q such that $\mathcal{K} \models Q$.*
***Output**: The set $\textsc{All-Just}(Q, \mathcal{K})$ of all the justifications for Q in $\mathcal{K}$.*

There are two categories of algorithms for finding a single justification: glass-box algorithms [9] and black-box algorithms. The former category is reasoner-dependent, i.e. a glass-box algorithm implementation depends on a specific reasoner, whereas a black-box algorithm is reasoner-independent, i.e. it can be used with any reasoner. In both cases, we still need a reasoner to obtain a justification.

It is possible to incrementally compute all justifications for an entailment by using Reiter's Hitting Set Tree (HST) algorithm [17]. This algorithm repeatedly calls a glass-box or a black-box algorithm which builds a new justification. To avoid the extraction of already found justifications, at each iteration the extraction process is performed on a KB from which some axioms are removed by taking into account the previously found justifications. For instance, given a KB $\mathcal{K}$ and a query Q, if the justification $\mathcal{J} = \{E_1, E_2, E_3\}$ was found, where E_is are axioms, to avoid the generation of the same justification, the HST algorithm tries to find a new justification on $\mathcal{K}' = \mathcal{K} \setminus E_1$. If no new justification is found the HST algorithm backtracks and tries to find another justification by removing other axioms from $\mathcal{J}$, one at a time.

4 Probabilistic Description Logics

DISPONTE [18,26] applies the distribution semantics [20] to Probabilistic Description Logic KBs.

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

$$p :: E$$

where $p \in [0,1]$ and E is a DL axiom. $p :: E$ means that we have degree of belief p in axiom E.

DISPONTE associates independent Boolean random variables to the DL axioms. The set of axioms that have the random variable assigned to 1 constitutes a *world*. The probability of a world w is computed by multiplying the probability p_i for each probabilistic axiom E_i included in the world by the probability $1-p_i$ for each probabilistic axiom E_i not included in the world.

Below, we provide some formal definitions for DISPONTE.

Definition 2 (Atomic choice). *An atomic choice is a couple (E_i, k) where E_i is the ith probabilistic axiom and $k \in \{0,1\}$. The variable k indicates whether E_i is chosen to be included in a world $(k = 1)$ or not $(k = 0)$.*

Definition 3 (Composite choice). *A composite choice κ is a consistent set of atomic choices, i.e., $(E_i, k) \in \kappa$, $(E_i, m) \in \kappa$ implies $k = m$ (only one decision is taken for each axiom).*

The probability of 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, because the random variables associated with axioms are independent.

Definition 4 (Selection). *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: $w_\sigma = \mathcal{C} \cup \{E_i | (E_i, 1) \in \sigma\}$, where $\mathcal{C}$ is the set of certain axioms.*

$P(w_\sigma)$ is a probability distribution over worlds. Let us indicate with $\mathcal{W}$ the set of all worlds. The probability of Q is [18]:

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

i.e. the probability of the query is the sum of the probabilities of the worlds in which the query is true.

Example 2. Let us consider the knowledge base and the query $Q = \mathsf{raskolnikov} : \mathsf{GreatMan}$ of Example 1 where some of the axioms are probabilistic:

$E_1 = 0.2 :: \mathsf{Nihilist} \sqsubseteq \mathsf{GreatMan}$ $\quad C_1 = \exists \mathsf{killed}.\top \sqsubseteq \mathsf{Nihilist}$

$E_2 = 0.6 :: (\mathsf{raskolnikov}, \mathsf{alyona}) : \mathsf{killed}$ $\quad E_3 = 0.7 :: (\mathsf{raskolnikov}, \mathsf{lizaveta}) : \mathsf{killed}$

Whoever is a nihilist is a "great man" with probability 0.2 (E_1) and Raskolnikov killed Alyona and Lizaveta with probability 0.6 and 0.7 respectively (E_2 and E_3). Moreover there is a certain axiom (C_1). The KB has eight worlds and Q is true in three of them, corresponding to the selections:

$$\{\{(E_1,1),(E_2,1),(E_3,1)\},\ \{(E_1,1),(E_2,1),(E_3,0)\},\ \{(E_1,1),(E_2,0),(E_3,1)\}\}$$

The probability is $P(Q) = 0.2{\cdot}0.6{\cdot}0.7{+}0.2{\cdot}0.6{\cdot}(1{-}0.7){+}0.2{\cdot}(1{-}0.6){\cdot}0.7 = 0.176$.

5 Inference in Probabilistic Description Logics

It is often infeasible to find all the worlds where the query is true. To reduce reasoning time, inference algorithms find, instead, explanations for the query and then compute the probability of the query from them. Below we provide the definitions of DISPONTE explanations and justifications, which are tightly intertwined with the previous definitions of explanation and justification for the non-probabilistic case.

Definition 5 (DISPONTE Explanation). *A composite choice ϕ identifies a set of worlds $\omega_\phi = \{w_\sigma | \sigma \in \mathcal{S}, \sigma \supseteq \phi\}$, where $\mathcal{S}$ is the set of all selections. We say that ϕ is an explanation for Q if Q is entailed by every world of ω_ϕ.*

Definition 6 (DISPONTE Justification). *We say that an explanation γ is a justification if, for all $\gamma' \subset \gamma$, γ' is not an explanation for Q.*

A set of explanations $\boldsymbol{\Phi}$ is *covering* Q if every world $w_\sigma \in \mathcal{W}$ in which Q is entailed is such that $w_\sigma \in \bigcup_{\phi \in \boldsymbol{\Phi}} \omega_\phi$. In other words a covering set $\boldsymbol{\Phi}$ identifies all the worlds in which Q succeeds.

Two composite choices κ_1 and κ_2 are *incompatible* if their union is inconsistent. For example, $\kappa_1 = \{(E_i, 1)\}$ and $\kappa_2 = \{(E_i, 0)\}$ are incompatible. A set $\boldsymbol{K}$ of composite choices is *pairwise incompatible* if for all $\kappa_1 \in \boldsymbol{K}$, $\kappa_2 \in \boldsymbol{K}$, $\kappa_1 \neq \kappa_2$ implies that κ_1 and κ_2 are incompatible. The *probability of a pairwise incompatible set of composite choices* K is $P(\boldsymbol{K}) = \sum_{\kappa \in \boldsymbol{K}} P(\kappa)$.

Given a query Q and a covering set of pairwise incompatible explanations $\boldsymbol{\Phi}$, the probability of Q is [18]:

$$P(Q) = \sum_{w_\sigma \in \omega_{\boldsymbol{\Phi}}} P(w_\sigma) = P(\omega_{\boldsymbol{\Phi}}) = P(\boldsymbol{\Phi}) = \sum_{\phi \in \boldsymbol{\Phi}} P(\phi) \tag{1}$$

where $\omega_{\boldsymbol{\Phi}}$ is the set of worlds identified by the set of explanations $\boldsymbol{\Phi}$.

Example 3. Consider the KB and the query $Q = \mathsf{raskolnikov} : \mathsf{GreatMan}$ of Example 2. We have the following covering set of pairwise incompatible explanations: $\boldsymbol{\Phi} = \{\{(E_1, 1), (E_2, 1)\}, \{(E_1, 1), (E_2, 0), (E_3, 1)\}\}$. The probability of the query is $P(Q) = 0.2 \cdot 0.6 + 0.2 \cdot 0.4 \cdot 0.7 = 0.176$.

Unfortunately, in general, explanations (and hence justifications) are not pairwise incompatible. The problem of calculating the probability of a query is therefore reduced to that of finding a covering set of justifications and then transforming it into a covering set of pairwise incompatible explanations.

We can think of using justification finding algorithms for non-probabilistic DLs to find the covering set of non-probabilistic justifications, then consider only the probabilistic axioms and transform the covering set of DISPONTE justifications into a pairwise incompatible covering set of explanations from which it is easy to compute the probability.

Example 4. Consider the KB and the query Q = raskolnikov : GreatMan of Example 2. If we use justification finding algorithms by ignoring the probabilistic annotations, we find the following non-probabilistic justifications: $\boldsymbol{\mathcal{J}} = \{\{E_1, C_1, E_2\},\ \{E_1, C_1, E_3\}\}$. Then we can translate them into DISPONTE justifications: $\boldsymbol{\Gamma} = \{\{(E_1, 1), (E_2, 1)\},\ \{(E_1, 1), (E_3, 1)\}\}$. Note that $\boldsymbol{\Gamma}$ is not pairwise incompatible, therefore we cannot directly use Eq. (1). The solution to this problem will be shown in the following section.

6 BUNDLE

The reasoner BUNDLE [18,19] computes the probability of a query w.r.t. DISPONTE KBs by first computing all the justifications for the query, then converting them into a pairwise incompatible covering set of explanations by building a Binary Decision Diagram (BDD). Finally, it computes the probability by traversing the BDD. A BDD for a function of Boolean variables is a rooted graph that has one level for each Boolean variable. A node n has two children corresponding respectively to the 1 value and the 0 value of the variable associated with the level of n. When drawing BDDs, the 0-branch is distinguished from the 1-branch by drawing it with a dashed line. The leaves store either 0 or 1.

Given the set $\boldsymbol{\Phi}$ of all DISPONTE explanations for a query Q, we can define the Disjunctive Normal Form Boolean formula $f_{\boldsymbol{\Phi}}$ representing the disjunction of all explanations as $f_{\boldsymbol{\Phi}}(\mathbf{X}) = \bigvee_{\phi \in \boldsymbol{\Phi}} \bigwedge_{(E_i,1)} X_i \bigwedge_{(E_i,0)} \overline{X_i}$. The variables $\mathbf{X} = \{X_i \mid p_i :: E_i \in \mathcal{K}\}$ are independent Boolean random variables with $P(X_i = 1) = p_i$ and the probability that $f_{\boldsymbol{\Phi}}(\mathbf{X})$ takes value 1 gives the probability of Q.

BDDs perform a Shannon's expansion of the Boolean function $f_{\boldsymbol{\Phi}}$ that makes the disjuncts, and hence the associated explanations, mutually exclusive, i.e. pairwise incompatible.

Given the BDD, we can use function Probability described in [10] to compute the probability. This dynamic programming algorithm traverses the diagram from the leaves to the root and computes the probability of a formula encoded as a BDD.

Example 5 (Example 2 cont.). Let us consider the KB and the query of Example 2. If we associate random variables X_1 with axiom E_1, X_2 with E_2 and X_3 with E_3, the BDD representing the set of explanations is shown in Fig. 1. By applying function Probability [10] to this BDD we get

$$\begin{aligned}
&\text{Probability}(n_3) = 0.7 \cdot 1 + 0.3 \cdot 0 = 0.7\\
&\text{Probability}(n_2) = 0.6 \cdot 1 + 0.4 \cdot 0.7 = 0.88\\
&\text{Probability}(n_1) = 0.2 \cdot 0.88 + 0.8 \cdot 0 = 0.176
\end{aligned}$$

and therefore $P(Q) = \text{Probability}(n_1) = 0.176$, which corresponds to the probability given by DISPONTE.

BUNDLE uses implementations of the HST algorithm to incrementally obtain all the justifications. However, *the first version was able to use only a glass-box approach which was dependent on the Pellet reasoner* [23].

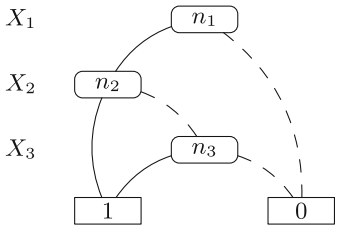

Fig. 1. BDD representing the set of explanations for the query of Example 2.

In the following, we illustrate the modifications introduced in the new modular version of BUNDLE.

Figure 2 shows the new architecture of BUNDLE. The main novelties are the adoption of the OWL Explanation library[1] [6] and of the BlackBox approach offered by OWL API. Thanks to them, BUNDLE is now reasoner-independent and it can exploit different OWL reasoners.

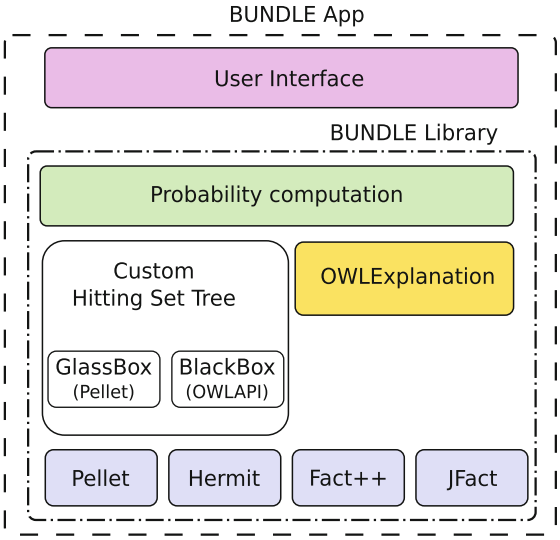

Fig. 2. Software architecture of BUNDLE.

Modularity is therefore realized in two directions: (1) support of different OWL reasoners: Pellet 2.5.0, Hermit 1.3.8.413 [22], Fact++ 1.6.5 [24], and JFact 4.0.4[2]; (2) three different strategies for finding a justification, which are:

[1] https://github.com/matthewhorridge/owlexplanation.
[2] http://jfact.sourceforge.net/.

GlassBox. A glass-box approach which depends on Pellet. It is a modified version of the `GlassBoxExplanation` class contained in the Pellet Explanation library.

BlackBox. A black-box approach offered by the OWL API[3] [5]. The OWL API is a Java API for the creation and manipulation of OWL 2 ontologies.

OWL Explanation. A library that is part of the OWL Explanation Workbench [6]. The latter also contains a Protégé plugin, underpinned by the library, that allows Protégé users to find justifications for entailments in their OWL 2 ontologies.

All reasoners can be paired with the BlackBox and OWL Explanation methods, while only Pellet can exploit the GlassBox method.

To find all justifications of a given query with the GlassBox and BlackBox approaches an implementation of the HST algorithm is used, which is a modified version of the `HSTExplanationGenerator` class of the OWL API. We modified this class in order to support annotated axioms (DISPONTE axioms are OWL axioms annotated with a probability). OWL Explanation, instead, already contains an HST implementation and a black-box approach that supports annotated axioms.

BUNDLE can be used as standalone desktop application or, in this new version, as a library.

6.1 Using BUNDLE as Application

BUNDLE is an open-source software and is available on Bitbucket, together with its manual, at https://bitbucket.org/machinelearningunife/bundle.

A BUNDLE image was deployed in Docker Hub. Users can start using BUNDLE with just a couple of docker commands. All they have to do is pull the image and start the container with the commands:

```
sudo docker pull giuseta/bundle:3.0.0
sudo docker run -it giuseta/bundle:3.0.0 bash
```

A bash shell of the container then starts and users can use BUNDLE by running the command `bundle`. For instance, if we consider the KB and the query of Example 2, the user can ask the query with:

```
bundle -instance http://www.semanticweb.org/
    crime_and_punishment#raskolnikov,http://www.
    semanticweb.org/crime_and_punishment#GreatMan file:
    examples/crime_and_punishment.owl
```

6.2 Using BUNDLE as Library

BUNDLE can also be used as a library. The library is set up as a Maven application and published on Maven Central[4].

[3] http://owlcs.github.io/owlapi/.

[4] With groupId `it.unife.endif.ml`, artifactId `bundle` and version `3.0.0`.

Once the developer has added BUNDLE dependency in the project's POM file, the probability of the query can be obtained in just few lines:

```
1 Bundle reasoner = new Bundle();
2 reasoner.setRootOntology(rootOntology);
3 reasoner.setReasonerFactory(new JFactFactory());
4 reasoner.init();
5 QueryResult result = reasoner.computeQuery(query);
```

where `rootOntology` and `query` are objects of the classes `OWLOntology` and `OWLAxiom` of the OWL API library respectively.

Line 3 shows that the developer can inject the preferred OWL API-based reasoner and perform probabilistic inference without modifying BUNDLE.

7 Experiments

We performed three different tests to compare the possible configurations of BUNDLE, which depend on the reasoner and the justification search strategy chosen, for a total of 9 combinations. In the first test we compared all configurations on four different datasets, in order to highlight which combination reasoner/strategy show the best behavior in terms of inference time. In the second one, we considered KBs of increasing size in terms of the number of probabilistic axioms. Finally, in the third test, we asked queries on a synthetic dataset of increasing size. The last two experiments were targeted to investigate the scalability of the different configurations. All tests were performed on the HPC System Marconi[5] equipped with Intel Xeon E5-2697 v4 (Broadwell) @ 2.30 GHz, using 8 cores for each test.

Test 1. The first test considers 4 real world KBs of various complexity as in [27]: (1) **BRCA** [11], which models the risk factors of breast cancer; (2) an extract of **DBPedia**[6] [13], containing structured information from Wikipedia, usually those contained in the information box on the righthand side of pages; (3) **Biopax level 3**[7] [3], which models metabolic pathways; (4) **Vicodi**[8] [16], which contains information on European history and models historical events and important personalities.

We used a version of the DBPedia and Biopax KBs without the ABox and a version of BRCA and Vicodi with an ABox containing 1 individual and 19 individuals respectively. For each KB we added a probability annotation to each axiom. The probability values were randomly assigned. We randomly created 50 subclass-of queries for all the KBs and 50 instance-of queries for BRCA and Vicodi, following the concepts hierarchy of the KBs, ensuring each query had at least one explanation.

[5] http://www.hpc.cineca.it/hardware/marconi.
[6] http://dbpedia.org/.
[7] http://www.biopax.org/.
[8] http://www.vicodi.org/.

Table 1 shows the average time in seconds to answer queries with different BUNDLE configurations. Bold values highlight the fastest configuration for each KB. With the exception of DBPedia, the best results are obtained by Pellet with the GlassBox approach, corresponding to the configuration of the previous non-modular version of BUNDLE. However, the use of OWL Explanation library with Pellet shows competitive results. For BioPax and Vicodi KBs, the BlackBox approach with Fact++ wasn't able to return a result (cells with "crash").

Table 1. Average time (in seconds) for probabilistic inference with all possible configurations of BUNDLE over different datasets (Test 1). For BioPax and Vicodi KBs Fact++/BlackBox wasn't able to return a result due to an internal error.

Reasoner	Method	Subclass-of queries				Instance-of queries	
		BioPax	DBPedia	BRCA	Vicodi	BRCA	Vicodi
Pellet	GlassBox	**0.501**	0.416	**0.85**	**0.393**	**1.654**	**0.42**
Pellet	BlackBox	1.779	0.484	1.488	0.667	5.671	0.804
Pellet	OWLExp	0.768	0.937	1.051	0.772	2.564	0.687
Hermit	BlackBox	4.281	2.192	7.68	1.968	29.944	2.416
Hermit	OWLExp	2.304	2.216	3.373	1.739	10.645	2.17
Fact++	BlackBox	crash	**0.254**	0.586	crash	3.368	crash
Fact++	OWLExp	1.568	1.077	0.934	0.667	2.532	1.183
JFact	BlackBox	1.757	0.501	1.974	0.726	7.273	0.812
JFact	OWLExp	1.072	1.248	2.036	0.869	3.47	1.291

Test 2. The second test was performed following the approach presented in [11] on the BRCA KB ($\mathcal{ALCHF}(D)$, 490 axioms). To test BUNDLE, we randomly generated and added an increasing number of subclass-of probabilistic axioms. The number of these axioms was varied from 9 to 16, and, for each number, 100 different consistent ontologies were created. Although the number of additional axioms, they may cause an exponential increase of the inference complexity (please see [11] for a detailed explanation).

Finally, an individual was added to every KB, randomly assigned to each simple class that appeared in the probabilistic axioms, and a random probability was attached to it. Complex classes contained in the conditional constraints were split into their components, e.g., the complex class *PostmenopausalWomanTakingTestosterone* was divided into *PostmenopausalWoman* and *WomanTakingTestosterone*. Finally, we ran 100 probabilistic queries of the form $a : C$ where a is the added individual and C is a class randomly selected among those that represent women under increased and lifetime risk such as *WomanUnderLifetimeBRCRisk* and *WomanUnderStronglyIncreasedBRCRisk*, which are at the top of the concept hierarchy.

Table 2 shows the execution time averaged over the 100 queries as a function of the number of probabilistic axioms. For each size, bold values indicate the best configuration. The BlackBox approaches are much slower on average than the others. The best performance is shown by Pellet/GlassBox until size 12, and by Pellet/OWLExp, Fact++/OWLExp and JFact/OWLExp from size 13.

Table 2. Average execution time (ms) for probabilistic inference with different configurations of BUNDLE on versions of the BRCA KB of increasing size (Test 2).

Reasoner	Method	9	10	11	12	13	14	15	16
Pellet	GlassBox	**1.360**	**1.076**	16.149	**16.448**	7.157	14.895	9.884	7.889
Pellet	BlackBox	19.747	16.406	51.258	42.258	42.309	65.690	63.269	55.006
Pellet	OWLExp	3.520	3.271	18.554	17.951	7.333	**8.848**	8.811	7.350
Hermit	BlackBox	31.871	26.373	72.473	62.064	66.664	77.378	79.785	57.745
Hermit	OWLExp	6.518	6.380	24.245	23.666	15.211	23.697	18.462	15.694
Fact++	BlackBox	3.718	2.846	20.880	18.879	8.479	19.221	15.837	10.411
Fact++	OWLExp	1.829	1.618	14.254	16.871	5.776	13.384	**8.224**	**6.640**
JFact	BlackBox	5.570	4.483	25.897	22.272	15.319	28.686	24.082	16.120
JFact	OWLExp	1.748	1.509	**13.366**	16.823	**2.267**	13.747	8.591	7.294

Test 3. In the third test we artificially created a set of KBs of increasing size of the following form:

$$(E_{1,i})\ 0.6 :: B_{i-1} \sqsubseteq P_i \sqcap Q_i \qquad (E_{2,i})\ 0.6 :: P_i \sqsubseteq B_i \qquad (E_{3,i})\ 0.6 :: Q_i \sqsubseteq B_i$$

where $n \geq 1$ and $1 \leq i \leq n$. The query $Q = B_0 \sqsubseteq B_n$ has 2^n explanations, even if the KB has a size that is linear in n.

We increased n from 2 to 10 in steps of 2 and we collected the running time, averaged over 50 executions. Table 3 shows, for each n, the average time in seconds that the systems took for computing the probability of the query Q (in bold the best time for each size). We set a timeout of 10 min for each query, so the cells with "–" indicate that the timeout occurred. This experiment confirms what already suggested by *Test 2*, i.e. the best results in terms of scalability are provided by the OWL Explanation method paired with any reasoner except Hermit. Thanks to this library the new version of BUNDLE is able to beat the first version (corresponding to Pellet/GlassBox), by reaching a larger dataset size.

Table 3. Average time (in seconds) for probabilistic inference with different configurations of BUNDLE on a synthetic dataset (Test 3). "–" means that the execution timed out (600 s).

Reasoner	Method	2	4	6	8	10
Pellet	GlassBox	0.404	**0.673**	**3.651**	–	–
Pellet	BlackBox	0.558	1.217	4.868	456.71	–
Pellet	OWLExp	0.972	1.957	4.45	**13.459**	**52.084**
Hermit	BlackBox	2.800	13.965	117.886	–	–
Hermit	OWLExp	2.307	8.507	37.902	185.158	–
Fact++	BlackBox	**0.248**	1.026	5.96	487.091	–
Fact++	OWLExp	0.815	1.708	4.282	15.331	76.313
JFact	BlackBox	0.405	1.178	4.895	497.745	–
JFact	OWLExp	0.946	1.878	4.258	17.547	78.831

8 Conclusions

In this paper, we presented a modular version of BUNDLE, a system for reasoning on Probabilistic Description Logics KBs that follow DISPONTE. Modularity allows one to pair 4 different OWL reasoners with 3 different approaches to find query justifications. In addition, BUNDLE can now be used both as a standalone application and as a library. We provided a comparison between the various configurations reasoner/approach over different datasets, showing that Pellet paired with GlassBox or any reasoner (except Hermit) paired with the OWLExplanation library achieve the best results in terms of inference time on a probabilistic ontology. In the future, we plan to study the effects of glass-box or grey-box methods for collecting explanations.

Acknowledgement. This work was supported by the "GNCS-INdAM".

References

1. Baader, F., Horrocks, I., Sattler, U.: Description logics, chap. 3, pp. 135–179. Elsevier, Amsterdam (2008)
2. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}^+$. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667, pp. 52–67. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74565-5_7
3. Demir, E., Cary, M.P., Paley, S., Fukuda, K., Lemer, C., Vastrik, I., Wu, G., D'Eustachio, P., Schaefer, C., Luciano, J.: The BioPax community standard for pathway data sharing. Nat. Biotechnol. **28**(9), 935–942 (2010)
4. Ding, Z., Peng, Y.: A probabilistic extension to ontology language OWL. In: 37th Hawaii International Conference on System Sciences (HICSS-37 2004), CD-ROM/Abstracts Proceedings, 5–8 January 2004, Big Island, HI, USA. IEEE Computer Society (2004)

5. Horridge, M., Bechhofer, S.: The OWL API: a Java API for OWL ontologies. Semant. Web **2**(1), 11–21 (2011)
6. Horridge, M., Parsia, B., Sattler, U.: The OWL explanation workbench: a toolkit for working with justifications for entailments in OWL ontologies (2009)
7. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, vol. 6, pp. 57–67. AAAI Press (2006). http://dl.acm.org/citation.cfm?id=3029947.3029959
8. Jaeger, M.: Probabilistic reasoning in terminological logics. In: Doyle, J., Sandewall, E., Torasso, P. (eds.) 4th International Conference on Principles of Knowledge Representation and Reasoning, pp. 305–316. Morgan Kaufmann (1994)
9. Kalyanpur, A.: Debugging and repair of OWL ontologies. Ph.D. thesis, The Graduate School of the University of Maryland (2006)
10. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. Theory Pract. Log. Prog. **11**(2–3), 235–262 (2011)
11. Klinov, P., Parsia, B.: Optimization and evaluation of reasoning in probabilistic description logic: towards a systematic approach. In: Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 213–228. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_14
12. Koller, D., Levy, A.Y., Pfeffer, A.: P-CLASSIC: a tractable probabilistic description logic. In: Kuipers, B., Webber, B.L. (eds.) Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 1997, 27–31 July 1997, Providence, Rhode Island, pp. 390–397. AAAI Press/The MIT Press (1997)
13. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia. Semant. Web **6**(2), 167–195 (2015)
14. Lukasiewicz, T.: Expressive probabilistic description logics. Artif. Intell. **172**(6–7), 852–883 (2008)
15. Lutz, C., Schröder, L.: Probabilistic description logics for subjective uncertainty. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) 12th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010), pp. 393–403. AAAI Press, Menlo Park (2010)
16. Nagypál, G., Deswarte, R., Oosthoek, J.: Applying the semantic web: the VICODI experience in creating visual contextualization for history. Lit. Linguist. Comput. **20**(3), 327–349 (2005)
17. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
18. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Probabilistic description logics under the distribution semantics. Semant. Web **6**(5), 447–501 (2015). https://doi.org/10.3233/SW-140154
19. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Reasoning with probabilistic ontologies. In: Yang, Q., Wooldridge, M. (eds.) 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 4310–4316. AAAI Press, Palo Alto (2015)
20. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995, pp. 715–729. MIT Press (1995)

21. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003, Acapulco, Mexico, 9–15 August 2003, pp. 355–362. Morgan Kaufmann Publishers Inc., San Francisco (2003)
22. Shearer, R., Motik, B., Horrocks, I.: HermiT: a highly-efficient OWL reasoner. In: OWL: Experiences and Direction, vol. 432, p. 91 (2008)
23. Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. J. Web Semant. **5**(2), 51–53 (2007)
24. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: system description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_26
25. W3C: OWL 2 web ontology language, December 2012. http://www.w3.org/TR/2012/REC-owl2-overview-20121211/
26. Zese, R.: Probabilistic semantic web: reasoning and learning, studies on the semantic web, vol. 28. IOS Press, Amsterdam (2017). https://doi.org/10.3233/978-1-61499-734-4-i, http://ebooks.iospress.nl/volume/probabilistic-semantic-web-reasoning-and-learning
27. Zese, R., Bellodi, E., Riguzzi, F., Cota, G., Lamma, E.: Tableau reasoning for description logics and its extension to probabilities. Ann. Math. Artif. Intell. **82**(1–3), 101–130 (2018). https://doi.org/10.1007/s10472-016-9529-3