Inference and Learning for Probabilistic Description Logics

Riccardo Zese
Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, 44122, Ferrara, Italy
riccardo.zese@unife.it

The last years have seen an exponential increase in the interest for the development of methods for combining probability with Description Logics (DLs). These methods are very useful to model real world domains, where incompleteness and uncertainty are common. This combination has become a fundamental component of the Semantic Web.

Our work started with the development of a probabilistic semantics for DL, called DISPONTE ("DIstribution Semantics for Probabilistic ONTologiEs", Spanish for "get ready"). DISPONTE applies the distribution semantics [Sato, 1995] to DLs. The distribution semantics is one of the most effective approaches in logic programming and is exploited by many languages, such as Independent Choice Logic, Probabilistic Horn Abduction, PRISM, pD, Logic Programs with Annotated Disjunctions, CP-logic, and ProbLog. Under DISPONTE we annotate axioms of a theory with a probability, that can be interpreted as an *epistemic probability*, i.e., as the degree of our belief in the corresponding axiom, and we assume that each axiom is independent of the others. DISPONTE, like the distribution semantics, defines a probability distribution over regular knowledge bases (also called worlds). To create a world, we decide whether to include or not each probabilistic axiom, then we multiply the probability of the choices done to compute the probability of the world. The probability of a query is then obtained from the joint probability of the worlds and the query by marginalization. Consider the Knowledge Base (KB) below:

$$0.5 \ :: \ \exists hasAnimal.Pet \sqsubseteq NatureLover \ (1)$$
$$0.6 \ :: \ Cat \sqsubseteq Pet \ (2)$$
$$tom : Cat \quad (kevin, tom) : hasAnimal$$
$$fluffy : Cat \quad (kevin, fluffy) : hasAnimal$$

It indicates that the individuals that own an animal which is a pet are nature lovers with a 50% probability and cats are pets with a 60% probability. Moreover, $kevin$ owns the animals $fluffy$ and tom which are both cats. The KB has four possible worlds: $\{\{(1),(2)\}, \ \{(1)\}, \ \{(2)\}, \ \{\}\}$ and the query axiom $Q = kevin : NatureLover$ is true in the first of them, while in the remaining ones it is false. The probability of the query is $P(Q) = 0.5 \cdot 0.6 = 0.3$.

Several algorithms have been proposed for supporting the development of the Semantic Web. Efficient DL reasoners, such us Pellet, RacerPro, and HermiT, are able to extract implicit information from the modeled ontologies. Despite the availability of many DL reasoners, the number of probabilistic reasoners is quite small, we can cite, for example, PRONTO, developed by the same authors of Pellet, which follows a semantics that defines probabilistic interpretations instead of a single probabilistic distribution over theories. In order to provide a tool able to manage DISPONTE probabilistic KBs, we developed BUNDLE, a reasoner based on Pellet [Sirin *et al.*, 2007] that allows to compute the probability of queries. BUNDLE, like most DL reasoners, exploits an imperative language for implementing its reasoning algorithm. This algorithm has to return the set of the explanations for the given query. An explanation is a subset of the KB which is sufficient for entailing the query. The explanations are then used for computing the probability of the query following the DISPONTE semantics. Nonetheless, usually reasoning algorithms use non-deterministic operators for doing inference. One of the most used approaches for doing reasoning is the tableau algorithm which applies a set of consistency preserving expansion rules to a $tableau$, that is an ABox. Some tableau expansion rules are non-deterministic, forcing the implementation of a search strategy in an or-branching search space. In addition, to compute all explanations for the given query, the exploration of all the non-deterministic choices done by the tableau algorithm is necessary.

In order to manage this non-determinism, we developed the system TRILL ("Tableau Reasoner for descrIption Logics in Prolog") which performs inference over DISPONTE DLs. It implements the tableau algorithm in the declarative Prolog language, whose search strategy is exploited for taking into account the non-determinism of the reasoning process. TRILL uses the Thea2[1] library for translating OWL KBs into Prolog facts. Both BUNDLE and TRILL implement the tableau algorithm for finding the set of explanations and use the inference techniques developed for probabilistic logic programs under the distribution semantics for computing the probability of the queries. They encode the explanations into a Boolean formula in Disjunctive Normal Form (DNF) in which each Boolean variable corresponds to an axiom of the KB. Then the formula is translated into a Binary Decision Diagram (BDD), from which the probability of the queries can be computed in time linear in the size of the diagram. A BDD for a Boolean formula is a rooted graph that has one level for each Boolean variable of the formula. Each

[1]http://www.semanticweb.gr/thea/

node represents a variable and has two children corresponding to the 1 value and to the 0 value of the variable. The leaves store either 1 or 0. We did several experiments for testing the performances of our reasoning algorithms. These tests show that Prolog is a viable language for implementing DL reasoning algorithms and that the performances of TRILL are comparable with those of a state-of-art reasoner, like BUNDLE. Encouraged by these results, we made TRILL available also via a web service, so that we can reach out to a wider audience and popularize the Probabilistic Semantic Web. The web service is called "TRILL on SWISH" and is available at http://trill.lamping.unife.it. We exploited SWISH [Lager and Wielemaker, 2014], a recently proposed web framework for logic programming that is based on various features and packages of SWI-Prolog. We modified it in order to manage OWL KBs. Moreover, we developed a second version of TRILL, called TRILL^P. Differently from TRILL and BUNDLE, which search for the set of explanations and translate it into a DNF formula, TRILL^P builds directly a monotone Boolean formula during the inference process, called "pinpointing formula" by Baader and Peñaloza [2010a; 2010b]. This formula compactly encodes the set of all explanations and can be directly translated into a BDD. In this way we avoid the step in which the set of explanations is translated into a DNF formula.

One of the problems of probabilistic KBs is that the parameters are difficult to set. It is thus necessary to develop systems which automatically learn the value of probabilities starting from the information available in the KB. For solving this problem, we presented EDGE ("Em over bDds for description loGics paramEter learning") that learns the parameters of a DISPONTE KB from the information available in the domain. It takes as input the KB and a set of examples of instances and non-instances of concepts and, for each example, executes BUNDLE for building the BDD representing its explanations. The parameters are then tuned using an Expectation-Maximization (EM) algorithm in which the required expectations are computed directly on the BDDs in an efficient way by repeatedly executing Expectation and Maximization steps until the log-likelihood of the examples reaches a local maximum. We compared EDGE with Goldminer[2] which extracts concept and role memberships information from a KB by means of SPARQL queries and exploits the apriori algorithm to find the relative Association Rules (ARs). Each AR is a probabilistic subclass axiom where the probability is the corresponding confidence. The comparison between the parameters learned by the two systems showed that EDGE can learn more accurate parameters than ARs. Starting from these results, we developed LEAP ("LEArning Probabilistic description logics"), an algorithm that combines EDGE with the learning system CELOE. The latter stands for "Class Expression Learning for Ontology Engineering" and is available in the Java open-source framework DL-Learner[3] for OWL and DLs. It provides a method for learning the structure of a KB which builds new (equivalence and subsumption) axioms. EDGE is used first to compute the initial log-likelihood of the (probabilistic) KB and then, during the execution of LEAP's learning algorithm, to learn the parameters of the refinements. For each axiom built by CELOE, LEAP creates a probabilistic subsumption axiom, where the probability is initialized to the accuracy returned by CELOE. Then LEAP adds the axioms one at a time to the KB and executes EDGE on the new KB to learn the new parameters and to choose whether these updates will be kept or discarded.

The main objective is to apply the developed algorithms to Big Data. In particular, the Semantic Web paved the way to the creation of *Linked Open Data*, where all the information is structured and linked together. Nonetheless, the size of the data requires the implementation of algorithms able to handle it. It is thus necessary to exploit approaches based on the parallelization and on cloud computing. Nowadays, we are working to improve EDGE in order to parallelize the process of building the BDDs and computing the expectations. The main idea is to divide the examples given to EDGE in chunks and assign one or more of them to each processor of each machine. Building a BDD for an example is independent from building it for each other examples. In this way, the number of examples we can handle increases and the execution time decreases. We also divide the expectation and the maximization phases so that the expectation step will be computed by slave processes, while the maximization step will be computed by the master process. The slave processes keep the BDDs in main memory through the iterations of the EM algorithm, execute the Expectation procedure and wait for the master, which sends a signal after the execution of the Maximization procedure that indicates whether to continue the EM cycle or not. The communication between the processes exploits the MPI protocol. We are implementing these modifications without altering the interface to EDGE. In this way, we can directly apply this new version to LEAP for reducing the running time spent for executing EDGE. Finally, we plan to test these new optimizations and also to study different scheduling approaches for assigning the examples to the slave processes, in order to choose which are the best and to evaluate the improvements.

References

[Baader and Peñaloza, 2010a] Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. *J. Autom. Reasoning*, 45(2):91–129, 2010.

[Baader and Peñaloza, 2010b] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *J. Log. Comput.*, 20(1):5–34, 2010.

[Lager and Wielemaker, 2014] Torbjörn Lager and Jan Wielemaker. Pengines: Web logic programming made easy. *TPLP*, 14(4-5):539–552, 2014.

[Sato, 1995] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of ICLP*, pages 715–729, Tokyo, 13-16 June 1995. MIT Press, Cambridge, MA.

[Sirin *et al.*, 2007] E. Sirin, B. Parsia, B. Cuenca-Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[2]https://code.google.com/p/gold-miner/

[3]http://dl-learner.org/Projects/DLLearner