SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 0000; **00**:1–19
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

A web system for reasoning with probabilistic OWL

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

[1] *Dipartimento di Ingegneria, University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy*
[2] *Dipartimento di Matematica e Informatica, University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy*

SUMMARY

We present the web application TRILL on SWISH, which allows the user to write probabilistic Description Logic (DL) theories and compute the probability of queries with just a web browser. Various probabilistic extensions of DLs have been proposed in the recent past, since uncertainty is a fundamental component of the Semantic Web. We consider probabilistic DL theories following our DISPONTE semantics. Axioms of a DISPONTE Knowledge Base (KB) can be annotated with a probability and the probability of queries can be computed with inference algorithms. TRILL is a probabilistic reasoner for DISPONTE KBs that is implemented in Prolog and exploits its backtracking facilities for handling the non-determinism of the tableau algorithm. TRILL on SWISH is based on SWISH, a recently proposed web framework for logic programming, based on various features and packages of SWI-Prolog (e.g., a web server and a library for creating remote Prolog engines and posing queries to them). TRILL on SWISH also allows users to cooperate in writing a probabilistic DL theory. It is free, open, and accessible on the Web at the url: `http://trill.lamping.unife.it`; it includes a number of examples that cover a wide range of domains and provide interesting Probabilistic Semantic Web applications. By building a web-based system, we allow users to experiment with Probabilistic DLs without the need to install a complex software stack. In this way we aim to reach out to a wider audience and popularize the Probabilistic Semantic Web. Copyright © 0000 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: semantic web; web applications; description logics; probabilistic description logics; SWI-Prolog; logic programming

1. INTRODUCTION

The main objective of the Semantic Web is modeling the real world in a way that is automatically understandable by machines [1]. To accomplish this objective, the W3C has standardized a family of knowledge representation formalisms, called Web Ontology Language (OWL). The first version of OWL defines three different sublanguages of increasing complexity: OWL-Lite, OWL-DL (based on Description Logics) and OWL-Full. Since the real world often contains uncertain information, it is of foremost importance to be able to represent and reason with such information. This problem has been studied by various authors both in the general case of First Order Logic (FOL) [2, 3, 4] and in the case of restricted logics, such as Description Logics (DLs) and Logic Programming (LP).

In particular, a field called Probabilistic Logic Programming (PLP) has recently arisen, which introduces probabilistic reasoning in logic programs. In this field, the distribution semantics [5] is

*Correspondence to: Riccardo Zese, Dipartimento di Ingegneria, University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy. E-mail: `riccardo.zese@unife.it` one of the most effective approaches and is exploited by many languages, such as Independent Choice Logic [6], Probabilistic Horn Abduction [7], PRISM [8], pD [9], Logic Programs with Annotated Disjunctions (LPADs) [10], CP-logic [11] and ProbLog [12]. The distribution semantics defines a probability distribution over normal logic programs called worlds and the probability of a query is obtained from this distribution by marginalization.

Following this line, in [13, 14, 15, 16] we defined DISPONTE ("DIstribution Semantics for Probabilistic ONTologiEs", Spanish for "get ready") which applies the distribution semantics to DLs by annotating axioms of a Knowledge Base (KB) with a probability and assuming that each axiom is independent of the others. A DISPONTE KB defines a probability distribution over regular KBs (also called worlds) and the probability of a query is obtained from the joint probability of the worlds and the query. This semantics paves the way for the Probabilistic Semantic Web.

Several algorithms have been proposed for supporting the development of the Semantic Web. Efficient DL reasoners, such us Pellet [17], RacerPro [18] and HermiT [19], are able to extract implicit information from ontologies, and probabilistic DL reasoners, such as PRONTO [20], are able to compute the probability of the inferred information. In this field, the tableau algorithm is most widespread approach. However, some tableau expansion rules are non-deterministic, forcing the implementation of a search strategy in an or-branching search space. In addition, in some cases we want to compute all explanations for a given query, requiring the exploration of all the non-deterministic choices made by the tableau algorithm during inference.

In order to manage this non-determinism, we developed the system TRILL ("Tableau Reasoner for descrIption Logics in proLog") [21, 22] which performs inference under DISPONTE. 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 library [23] for translating OWL KBs into Prolog facts and exploits Binary Decision Diagrams for computing the probability of queries from the set of all explanations, in a time that is linear in the size of the diagram.

In this paper, we present the "TRILL on SWISH" web application for performing inference on user-defined KBs under DISPONTE. It is based on the SWISH web framework [24], which exploits features and packages of SWI-Prolog and its Pengines library. TRILL on SWISH allows the user to write a KB in OWL, using the RDF/XML syntax, and ask a query to it. Then, using JavaScript, the query and the program are sent to the server that builds a Pengine (Prolog Engine), which translates the KB, evaluates the query and returns answers for it. TRILL on SWISH can answer different types of queries and can compute the probability of the query, whether the query is entailed or not from the knowledge base, and the set of explanations that explain the entailment. Both the web server and the inference back-end are run entirely within SWI-Prolog. Reasoning in TRILL on SWISH is accomplished by a version of TRILL ported to SWI-Prolog. We also modified SWISH both in its server and client parts. TRILL on SWISH is available at `http://trill.lamping.unife.it`. Its interface is shown in Figure 1 and offers several examples of KBs and queries.

Prolog is a viable language for implementing DL reasoning algorithms and it is emerging as a valid tool for the Semantic Web [25]. TRILL on SWISH allows users to experiment with these algorithms without the need to install a system, a procedure which is often complex, error prone and limited mainly to the GNU/Linux platform. In addition SWISH supports real-time collaboration, a useful feature for both development and educational purposes.

The paper is organized as follows. Section 2 briefly introduces DLs and Section 3 presents DISPONTE. Section 4 illustrates the TRILL algorithm, Section 5 describes the SWISH web platform and Section 6 the integration of TRILL in it. Finally, Section 7 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 [26, 27] for excellent introductions. DLs

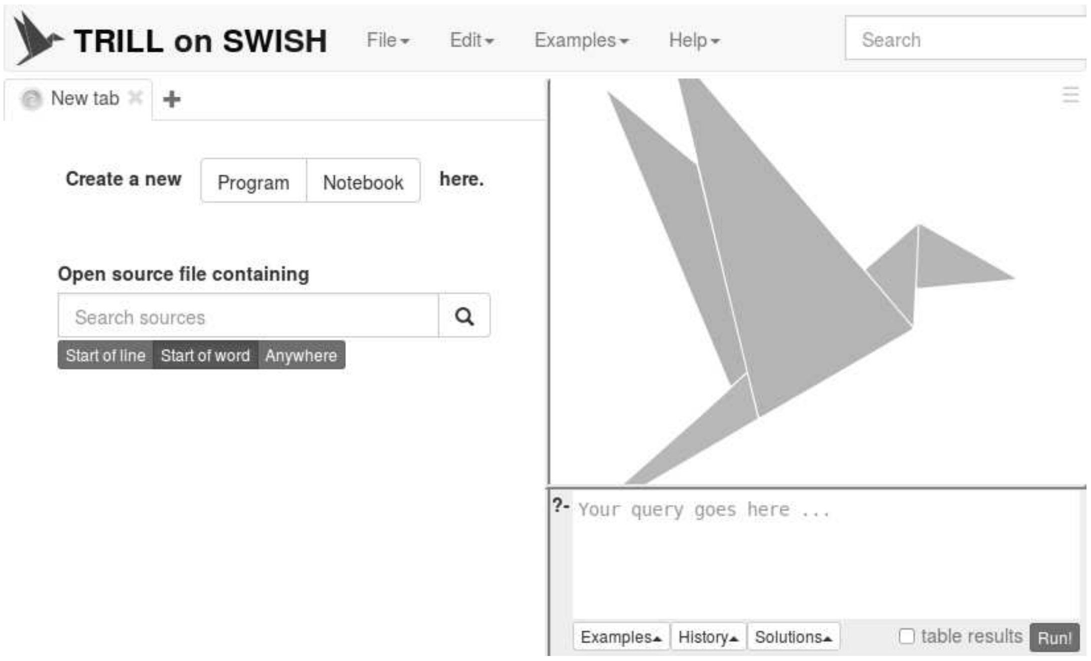


Figure 1. "TRILL on SWISH" web interface.

are particularly useful for representing ontologies and have been adopted as the basis of the Semantic Web.

While DLs can be translated into FOL, 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 corresponds to a set of pairs of individuals of the domain. In order to illustrate DLs, we now describe $\mathcal{SHOIN}(\mathbf{D})$, the basis of OWL DL.

Let $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ be sets of *atomic concepts*, *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 by induction as follows. Each $C \in \mathbf{A}$, $\bot$ and $\top$ are concepts. If $a \in \mathbf{I}$, then $\{a\}$ is a concept called *nominal*, and 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 $\exists R.C$, $\forall R.C$, $\geq nR$ 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 the conjunction of $C \sqsubseteq D$ 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 knowledge base $\mathcal{K}$ is usually 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 $C \in \mathbf{A}$ 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:

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

$\mathcal{SHOIN}(\mathbf{D})$ allows the definition of 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 $\mathcal{K}$.

Inference in a DL is *decidable* if the problem of checking the satisfiability of any possible KB representable with that DL 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 Q over a KB $\mathcal{K}$ is usually an axiom for which we want to test the entailment from the KB, written $\mathcal{K} \models Q$. The entailment test in $\mathcal{SHOIN}(\mathbf{D})$ may be reduced to checking the unsatisfiability of a concept in the knowledge base, 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 $C \sqcap \neg D$.

Example 1
The following KB is derived from the ontology `people+pets` [28]:

$$
\begin{array}{l}
\exists hasAnimal.Pet \sqsubseteq NatureLover \\
\mathit{fluffy} : Cat \\
tom : Cat \\
Cat \sqsubseteq Pet \\
(kevin, \mathit{fluffy}) : hasAnimal \\
(kevin, tom) : hasAnimal
\end{array}
$$

It states that individuals that own an animal which is a pet are nature lovers and that $kevin$ owns the animals *fluffy* and tom. Moreover, *fluffy* and tom are cats and cats are pets. The query $Q = kevin : NatureLover$ is entailed by the KB.

3. DISPONTE

DISPONTE [29] applies the distribution semantics [5] of probabilistic logic programming to DLs. A program following this semantics defines a probability distribution over *worlds* that are normal logic programs. Then the distribution is extended to a joint distribution over the query and the programs from which the probability of the query is obtained by marginalization.

In DISPONTE, a *probabilistic knowledge base* $\mathcal{K}$ contains a set of certain axioms and a set of *probabilistic axioms* which take the form

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

where p is a real number in $[0, 1]$ and E is any DL axiom of the KB (either of the terminological or assertional part of the KB).

The idea of DISPONTE is to associate independent Boolean random variables to the probabilistic axioms. To obtain a *world* w we decide whether to include each probabilistic axiom or not in w by assigning values to every random variable. A world is the set of axioms whose random variables are assigned the value 1. A world is therefore 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 $C(a)$. A probabilistic concept inclusion axiom of the form $p :: C \sqsubseteq D$ represents the fact that we believe in the truth of $C \sqsubseteq D$ with probability p. For example, the axioms

$$\begin{array}{lll} 0.9 & :: & tweety : Flies \\ 0.7 & :: & Bird \sqsubseteq Flies \end{array}$$

mean that Tweety flies with probability 0.9 and that we believe in the fact that birds fly with probability 0.7.

Formally, an *atomic choice* is a pair (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 *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 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 axiom of the theory. A selection σ identifies the theory, also called *world*, $w_\sigma = \{E_i | (E_i, 1) \in \sigma\}$. Let us indicate 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 defined by marginalizing the joint probability of the query and the worlds:

$$\begin{aligned} P(Q) &= \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q, w) && (2) \\ &= \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q|w)P(w) && (3) \\ &= \sum_{w\in\mathcal{W}_\mathcal{K}: w\models Q} P(w) && (4) \end{aligned}$$

where (2) and (3) follow for the sum and product rules of probability theory respectively and (4) holds because $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise.

Example 2
Consider the KB presented in Example 1 where we turned some axioms into probabilistic axioms:

$$\begin{array}{lr} \exists hasAnimal.Pet \sqsubseteq NatureLover & \\ (kevin, fluffy) : hasAnimal & \\ (kevin, tom) : hasAnimal & \\ 0.4 :: fluffy : Cat & \text{E1} \\ 0.3 :: tom : Cat & \text{E2} \\ 0.6 :: Cat \sqsubseteq Pet & \text{E3} \end{array}$$

The KB indicates that the individuals that fluffy and tom are cats with 40% and 30% probability respectively and cats are pets with a 60% probability. The KB has eight possible worlds, corresponding to the selections:

$$\begin{array}{ll} \{(\mathrm{E1},0),(\mathrm{E2},0),(\mathrm{E3},0)\} & \{(\mathrm{E1},1),(\mathrm{E2},0),(\mathrm{E3},0)\} \\ \{(\mathrm{E1},0),(\mathrm{E2},0),(\mathrm{E3},1)\} & \{(\mathrm{E1},1),(\mathrm{E2},0),(\mathrm{E3},1)\} \\ \{(\mathrm{E1},0),(\mathrm{E2},1),(\mathrm{E3},0)\} & \{(\mathrm{E1},1),(\mathrm{E2},1),(\mathrm{E3},0)\} \\ \{(\mathrm{E1},0),(\mathrm{E2},1),(\mathrm{E3},1)\} & \{(\mathrm{E1},1),(\mathrm{E2},1),(\mathrm{E3},1)\} \end{array}$$

and the query axiom $Q = kevin : NatureLover$ is true in three of them, corresponding to the following choices:

$$\{(\mathrm{E1},0),(\mathrm{E2},1),(\mathrm{E3},1)\}$$
$$\{(\mathrm{E1},1),(\mathrm{E2},0),(\mathrm{E3},1)\}$$
$$\{(\mathrm{E1},1),(\mathrm{E2},1),(\mathrm{E3},1)\}$$

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

Note that a DISPONTE KB with inconsistent worlds should not be used to derive consequences, just as a regular DL KB that is inconsistent should not.

A different approach for assigning a semantics to probabilistic DLs is followed in [30] and [31]. Here, a KB is associated with a Bayesian network with variables V. Axioms take the form $E : X = x$ where E is a DL axiom and $X = x$ is an annotation with $X \subseteq V$ and x a set of values for these variables. The Bayesian network assigns a probability to every assignment of V, called a world. The authors show that the probability of a query $Q = E : X = x$ is given by the sum of the probabilities of the worlds where $X = x$ is satisfied and where E is a logical consequence of the theory composed of the annotated axioms whose annotation is true in the world. DISPONTE is a special case of this semantics where every axiom $E_i : X_i = x_i$ is such that X_i is a single Boolean variable and the Bayesian network has no edges, i.e., all the variables are independent. This is an important special case that greatly simplifies reasoning, as computing the probability of the worlds takes a time linear in the number of variables.

4. TRILL

TRILL ("Tableau Reasoner for descrIption Logics in Prolog") [21, 22] implements a tableau algorithm in Prolog. It is able to compute the set of all the explanations of queries w.r.t. both probabilistic and non-probabilistic KBs. Moreover, it can compute the probability of queries w.r.t. probabilistic KBs. In this case, after generating the explanations, TRILL converts them into a Binary Decision Diagram (BDD) which is exploited to efficiently compute the probability of the query. TRILL can answer concept membership queries and subsumption queries, and can find explanations both for the unsatisfiability of a concept contained in the KB or for the inconsistency of the entire KB. TRILL is implemented in Prolog, so the management of the rules' non-determinism is delegated to this language.

For converting OWL DL KBs into Prolog, we use the Thea2 library [23]. Thea2 performs a direct translation of the OWL axioms into Prolog facts. For example, a simple subclass axiom between two named classes $Cat \sqsubseteq Pet$ is written using the `subClassOf/2` predicate as `subClassOf('Cat','Pet')`. For more complex axioms, Thea2 exploits the *list* Prolog construct, so the axiom

$$NatureLover \equiv PetOwner \sqcup GardenOwner$$

becomes

```
equivalentClasses(['NatureLover',
unionOf(['PetOwner','GardenOwner'])]).
```

Table I. Comparison between an OWL axiom and its Prolog translation.

OWL	$forebrain\_neuron \equiv neuron \sqcap \exists partOf.forebrain$
Prolog	`equivalentClasses( [forebrain_neuron, intersectionOf([neuron, someValuesFrom(partOf, forebrain)])])`

When a probabilistic KB is given as input to TRILL, for each probabilistic axiom of the form $Prob :: Axiom$, two facts are asserted, the axiom itself and an annotation assertion of the form `annotationAssertion( ProbAnnot, Axiom, literal(Prob))`, where *ProbAnnot* is the name of the annotation, $Axiom$ is the probabilistic axiom and $Prob$ is the probability value. Complex classes are represented by means of function symbols, as shown in Table I.

A *tableau* is an ABox. It can also be seen as a graph G 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 pair (a, b) belongs. A tableau algorithm proves an axiom by refutation: it starts from a tableau that contains the negation of the axiom and applies the tableau expansion rules. For example, if the query is a class assertion, $C(a)$, we add $\neg C$ to the label of a. A *tableau 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. A clash is, for example, a concept C and a node a where C and $\neg C$ are present in its label, i.e. $\{C, \neg C\} \subseteq \mathcal{L}(a)$. If no clashes are found, the tableau represents a model for the negation of the query, which is thus not entailed. Each expansion rule updates as well a *tracing function* τ, which associates sets of axioms with labels of nodes and edges. It maps pairs (concept, individual) or (role, pair of individuals) to a fragment of the knowledge base $\mathcal{K}$. τ is initialized to the empty set for all the domain elements, 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 tableau algorithm output is a set S of axioms that is a fragment of $\mathcal{K}$ from which the query is entailed.

In [32] the authors showed that pinpointing extensions of tableau algorithms need a *blocking condition* in order to ensure termination. TRILL uses the blocking condition described in [33]. In a tableau a node x can be a *nominal* node if its label $\mathcal{L}(x)$ contains a *nominal*, or a *blockable* node otherwise. If there is an edge $e = \langle x, y\rangle$ then y is a *successor* of x and x is a *predecessor* of y. *Ancestor* is the transitive closure of predecessor while *descendant* is the transitive closure of successor. A node y is called *R-neighbour* of a node x if y is a successor of x and $R \in \mathcal{L}(\langle x, y\rangle)$, where $R \in \mathbf{R}$.

An R-neighbour y of x is *safe* if (i) x is blockable or if (ii) x is a nominal node and y is not blocked. Finally, a node x is *blocked* if it has ancestors x_0, y and y_0 such that all the following conditions are true: (1) x is a successor of x_0 and y is a successor of y_0, (2) y, x and all nodes on the path from y to x are blockable, (3) $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x_0) = \mathcal{L}(y_0)$, (4) $\mathcal{L}(\langle x_0, x\rangle) = \mathcal{L}(\langle y_0, y\rangle)$. In this case, we say that y *blocks* x. A node is blocked also if it is blockable and all its predecessors are blocked; if the predecessor of a safe node x is blocked, then we say that x is *indirectly blocked*.

In order to represent the tableau, TRILL uses a pair *Tableau = A-T*, where A is a list containing information about nominal individuals and class and role assertions with the corresponding explanation, while T is a fact `tnr(G,RBN,RBR)` in which `G` is a directed graph that contains the main structure of the tableau, `RBN` is a red-black tree (a key-value dictionary) in which a key is a pair of individuals and its value is the set of the labels of the edge between the two individuals, and `RBR` is a red-black tree in which a key is a role and its value is the set of pairs of individuals that are linked by the role. This representation allows TRILL to quickly find the information needed during the execution of the tableau algorithm. For managing the *blocking* system we use a predicate for each blocking state: `nominal/2`, `blockable/2`, `blocked/2`, `indirectly_blocked/2` and `safe/3`. Each predicate takes as arguments the individual *Ind* and the tableau *A-T*; `safe/3` takes as input also the role R. For each individual `ind` in the ABox we add the atom `nominal(ind)` to A, then every time we have to check the blocking status of an individual we call the corresponding predicate that returns the status by checking the tableau. For example, `indirectly_blocked/2` is implemented as:

```
indirectly_blocked(Ind,ABox-tnr(T,RBN,RBR)):-
  transpose(T,T1),
  neighbours(Ind,T1,N),
  member(A,N),
  blocked(A,ABox-tnr(T,RBN,RBR)),!.
```

where `transpose/2` builds a transposed version of the tableau, which is examined by `neighbours/3` that returns in `N` the list of neighbours of `Ind`. Then, `member/2` and `blocked/2` are exploited to check if there is at least an individual in `N` which is blocked.

Deterministic and non-deterministic tableau expansion rules are treated differently in TRILL. *Non-deterministic* rules are implemented by a predicate `<rule_name>(Tab0, TabList)` that, given the current tableau `Tab0`, returns the list of tableaux `TabList` created by the application of the rule to `Tab0`. For example, the non-deterministic rule that expands the concept $A \sqcup B$ is implemented by the following code:

```
or_rule(ABox0-Tabs0,L):-
  find((classAssertion(unionOf(LC),Ind),Expl),ABox0),
  \+indirectly_blocked(Ind,ABox0-Tabs0),
  findall(ABox1-Tabs0,scan_or_list(LC,Ind,
         Expl,ABox0,Tabs0,ABox1),L),
  dif(L,[]),!.

scan_or_list([],_Ind,_Expl,ABox,_Tabs,ABox).

scan_or_list([C|_T],Ind,Expl,ABox,Tabs,
  [(classAssertion(C,Ind),Expl)|ABox]):-
  absent(classAssertion(C,Ind),Expl,ABox-Tabs).

scan_or_list([_C|T],Ind,Expl,ABox0,Tabs,ABox):-
   scan_or_list(T,Ind,Expl,ABox0,Tabs,ABox).
```

The predicate `or_rule/2` searches `Tab0`, which corresponds to the pair `ABox0-Tabs0`, for an individual to which the rule can be applied and returns in `L` the list of new tableaux created by `scan_or_list/6`.

Deterministic rules are implemented by a predicate `<rule_name>(Tab0,Tab)` that, given the current tableau `Tab0`, returns the tableau `Tab` obtained by the application of the rule to `Tab0`. For example, the *unfold* rule looks for subclass and class equivalence axioms in order to add information to individuals. A snippet of the code which corresponds to this rule is reported here:

```
unfold_rule(ABox0-Tabs0,
                 [(classAssertion(D,Ind),[Ax|Expl])|ABox]-Tabs0):-
  find((classAssertion(C,Ind),Expl),ABox0),
  find_sub_sup_class(C,D,Ax),
  absent(classAssertion(D,Ind),[Ax|Expl],ABox0-Tabs0),
  add_nominal(D,Ind,ABox0,ABox).

find_sub_sup_class(C,D,subClassOf(C,D)):-
  subClassOf(C,D).

find_sub_sup_class(C,D,equivalentClasses(L)):-
  equivalentClasses(L),
  member(C,L),
  member(D,L),
  dif(C,D).
```

The `unfold_rule/2` predicate searches `ABox0-Tabs0`, corresponding to `Tab0`, for an individual to which the rule can be applied and calls the `find_sub_sup_class/3` predicate in order to find the class to be added to the individual. `find/2` implements the search for a class assertion. Since the data structure that stores class assertions is currently a list, `find/2` simply calls `member/2`. `absent/3` checks if the class assertion axiom with the associated explanation is already present in the tableau, while `add_nominal/4` handles nominal individuals in case `D` is a nominal concept.

Expansion rules are applied in order by `apply_all_rules/2`, first the non-deterministic ones and then the deterministic ones. The `apply_nondet_rules(RuleList, Tab0, Tab)` predicate takes as input the list of non-deterministic rules and the current tableau and returns a tableau obtained by the application of one of the rules. `apply_nondet_rules/3` is called as `apply_nondet_rules( [or_rule, max_rule], Tab0, Tab)` and is shown below:

```
apply_all_rules(Tab0,Tab):-
  apply_nondet_rules([or_rule,max_rule],Tab0,Tab1),
  (Tab0=Tab1 ->
    Tab=Tab1
  ;
    apply_all_rules(Tab1,Tab)
  ).

apply_nondet_rules([],Tab0,Tab):-
  apply_det_rules([o_rule,and_rule,
  unfold_rule,add_exists_rule,
  forall_rule,forall_plus_rule,
  exists_rule,min_rule],Tab0,Tab).

apply_nondet_rules([H|T],Tab0,Tab):-
  once(call(H,Tab0,L)),
  member(Tab,L),
  dif(Tab0,Tab).

apply_nondet_rules([_|T],Tab0,Tab):-
  apply_nondet_rules(T,Tab0,Tab).
```

If a non-deterministic rule is applicable, the tableau list obtained by its application is returned by the predicate corresponding to the applied rule, a cut is performed to avoid backtracking to other rule choices and a tableau from the list is non-deterministically chosen with the `member/2` predicate.

If no non-deterministic rule is applicable, deterministic rules are tried sequentially by the predicate `apply_det_rules/3`, that is called as `apply_det_rules(RuleList,Tab0,Tab)`. It takes as input the list of deterministic rules and the current tableau and returns a tableau obtained by the application of one of the rules.

```
apply_det_rules([],Tab,Tab).

apply_det_rules([H|_],Tab0,Tab):-
  once(call(H,Tab0,Tab)).

apply_det_rules([_|T],Tab0,Tab):-
  apply_det_rules(T,Tab0,Tab).
```

After the application of a deterministic rule, a cut avoids backtracking to other possible choices for the deterministic rules. If no rule is applicable, the input tableau is returned and rule application stops, otherwise a new round of rule application is performed.

Once the set of explanations K for the query Q is found, we can define the Disjunctive Normal Form (DNF) Boolean formula ϕ_K in this way

$$\phi_K = \bigvee_{\kappa \in K} \bigwedge_{(E_i,1)\in\kappa} var(E_i).$$

where $var(E_i)$ is the propositional variable associated with the probabilistic axiom E_i. ϕ_K is also called *pinpointing formula*[†] and it is easy to see that every set of propositional variables that makes ϕ_K true uniquely corresponds to a world where Q is true.

[†]It corresponds to the *clash formula* introduced in [34].

Computing the probability of a DNF formula is a #P-hard problem [35], even if all variables are independent, as they are in our case. An approach that seems to give good results in practice is *knowledge compilation* [36]. TRILL applies it by translating ϕ_K into a BDD. The size of the obtained BDD depends on the number of probabilistic axioms used for the explanations rather than on the logic. The problem of compiling a Boolean formula into the smallest BDD is NP-hard [37]. Therefore we cannot hope to reduce the complexity in the worst case, we only hope to be able to handle non-trivial average cases. Finally, TRILL computes the probability of the query from the BDD in polynomial time [21].

TRILL is available for Yap Prolog[‡] [38] and SWI-Prolog[§] [39], which is the basis of the TRILL on SWISH web application described in Section 6.

5. SWISH

SWISH[¶] is a web application based on SWI-Prolog that allows the users to write Prolog programs and ask queries through the browser without installing anything on their machines. The SWISH page, shown in Figure 1, is divided into three panels: the left one contains a program editor, the bottom right the query editor and the top right the query results. The query editor contains also the button “Run!”, which creates a JavaScript object called *runner*. The runner collects the text in the program editor and the query and sends this information to the server, which creates a Pengine (Prolog Engine). The Pengine initializes a temporary private module in which the program is compiled, then it checks whether the query execution is safe. If executing the query may compromise the system, an error is returned, otherwise the query is computed and the results are returned to the runner (and thus to the user) through JSON messages.

SWISH uses the SWI-Prolog Pengines library [24], which allows to create Prolog engines from different sources: (1) an ordinary Prolog thread, (2) another Pengine, (3) JavaScript running in a web client. Each Pengine is associated with a Prolog thread with a message queue for incoming requests, a message queue for outgoing responses and a dynamic clause database, all private to the Pengine.

The Pengine library follows a master/slave architecture. The master creates a Pengine on the slave and sends a query to it. The conversations between them follow the Prolog Transport Protocol (PLTP), a protocol based on HTTP.

Example 3
Consider this example from [24]:

```
:- use_module(library(pengines)).
main :-
    pengine_create([
        server('http://pengines.org'),
        src_text("
            q(X) :- p(X).
            p(a). p(b). p(c).
        ")
    ]),
    pengine_event_loop(handle, []).

handle(create(ID, _)) :-
    pengine_ask(ID, q(X), []).
```

[‡]http://www.dcc.fc.up.pt/~vsc/Yap/
[§]http://www.swi-prolog.org/
[¶]http://swish.swi-prolog.org/

```
handle(success(ID, [X], false)) :-
    writeln(X).
handle(success(ID, [X], true)) :-
    writeln(X),
    pengine_next(ID, []).
```

This code calls `pengine_create/1` to create a slave Pengine in a remote Pengine server and uses the `src_text` option to send a textual message to the Pengine containing the program to be queried. `pengine_event_loop/2` starts a listener loop which waits for event terms. Once it finds an event, it calls `handle/1` on it. `handle/1` can manage two different events:

1. `create(ID,_)`: the Pengine identified by `ID` has been created and the event handler uses `pengine_ask/3` to ask a query; `pengine_ask/3` is deterministic and the results of the query will be returned in the form of event terms;
2. `success(ID, Inst, More)`: the Pengine identified by `ID` has successfully solved the query, `Inst` holds instantiations for the variables of the query and `More` is either `true` or `false`, indicating whether we may expect the Pengine to be able to return more solutions or not; `pengine_next/2` is called if `More` is true, to get the next solution.

Thus, by running `main/0`, we will see the terms `q(a)`, `q(b)` and `q(c)` written to standard output.

Code sent to Pengines is executed in a "sandboxed" environment ensuring that only predicates that do not have side effects - such as accessing the file system, loading foreign extensions, defining other predicates outside the sandboxed environment, etc. - are called.

SWI-Prolog also offers a JavaScript library called `pengine.js` that allows the creation of Pengine JavaScript objects on the server and to query them from JavaScript.

The SWISH web server is implemented by the SWI-Prolog HTTP package, a series of libraries for serving data on HTTP [40].

SWISH allows users to collaborate on code development. It exploits the TogetherJS[||] library, that is an open source JavaScript library built and hosted by Mozilla. TogetherJS offers different built-in features which permit a real time interaction between users:

Audio and Text Chat The collaborators can chat by talking or texting to each other.

User Focus The collaborators see each other's mouse cursors and clicks.

Co-browsing The collaborators can follow each other to different pages on the same domain.

Real time content sync The content is synchronized between all the collaborators.

It is possible to start collaborating on SWISH by clicking the item "File" in the menu bar and then clicking on "Collaborate..". The TogetherJS dock will appear and you can invite other users by sharing the generated link.

For TRILL on SWISH we used the version of SWISH included in ClioPatria[**], a Semantic Web server based on SWI-Prolog. This version was chosen because it offers the RDF handling features of ClioPatria.

6. TRILL ON SWISH

In order to implement TRILL on SWISH, we installed the `cplint` pack [41], which includes a foreign language C library for building BDDs, and the `trill` pack into SWI-Prolog. `cplint`

[||]https://togetherjs.com/

[**]`http://cliopatria.swi-prolog.org/home`

can be installed by the user with the command `pack_install(cplint)` at the SWI-Prolog prompt. Similarly, `trill` can be installed with the command `pack_install(trill)`. After loading TRILL with `use_module(library(trill))`, the KB must be loaded in memory. This is done by exploiting Thea2.

TRILL on SWISH, whose interface is shown in Figure 1, allows the user to write a KB in the RDF/XML format in the left panel and write a query in the bottom right panel. Both the KB and query editor have syntax highlighting. Moreover, URIs in queries can be written without the base URI or using a namespace defined in the RDF/XML file and the system checks for possible misspellings of URIs that are reported to the user.

In case one needs KB serializations different from RDF/XML or prefers a GUI to build the KB, it is possible to use WebProtégé [42] to develop the KB, then download it in RDF/XML and upload it into TRILL on SWISH.

Currently, the queries that can be executed are specified using the Prolog syntax. The types of query that are available are shown below:

1. `sub_class(Class1,Class2)`,
   `sub_class(Class1,Class2,Expl)`,
   `prob_sub_class(Class1,Class2,Prob)`,
2. `instanceOf(Class,Individual)`,
   `instanceOf(Class,Individual,Expl)`,
   `prob_instanceOf(Class,Individual,Prob)`,
3. `unsat(ClassExpression)`,
   `unsat(ClassExpression,Expl)`,
   `prob_unsat(ClassExpression,Prob)`,
4. `inconsistent_theory`,
   `inconsistent_theory(Expl)`,
   `prob_inconsistent_theory(Prob)`.

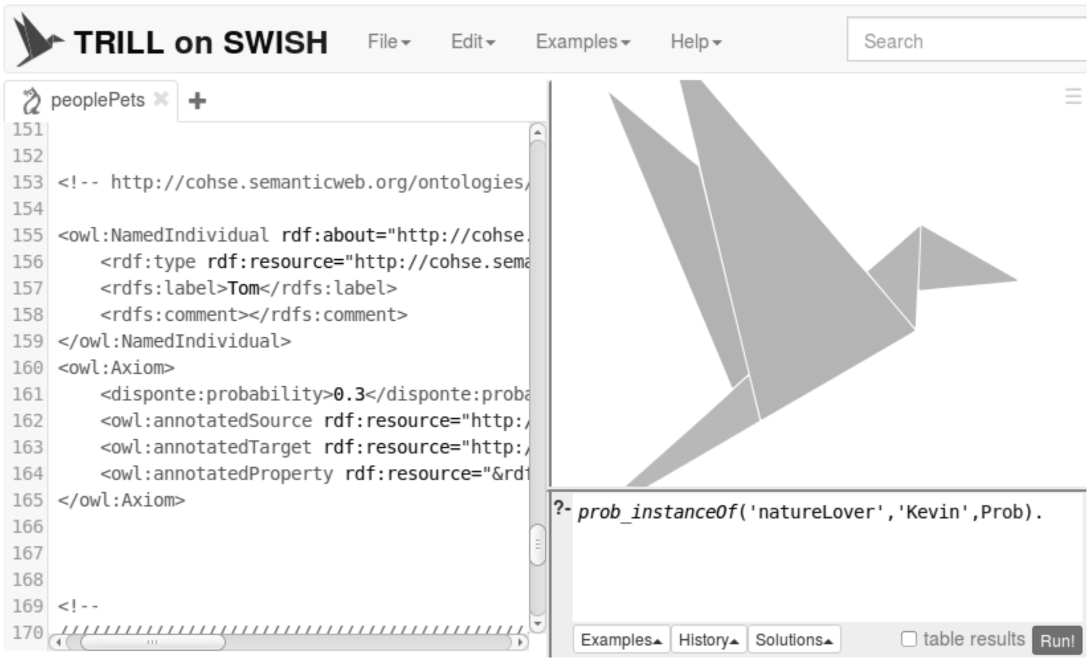
Each query has three versions which respectively test, find an explanation `Expl` or compute the probability `Prob` of the query. Once the query is written (Figure 2), by pressing the "Run!" button the information is sent to the server. This in turn creates a Pengine with the program. The creation of a new Pengine object is done by the `runner.js` JavaScript file. `runner.js` was modified by adding directives for loading the TRILL library, for disabling the check for discontiguous clauses, and for parsing RDF/XML and translating it into Prolog. This is done by the following snippet of `runner.js`:

```
data.prolog = new Pengine({
  ...
  src: ":-use_module(library(trill)).
    :-use_module(library(translate_rdf)).
    :-use_module(library(pengines)).

    parse:- pengine_self(M),
    set_prolog_flag(M:unknwon,fail),
    query.source+"').",
  ...
  oncreate: handleCreate,
  ...
  });
```

that stores a new Pengine object in the runner's `data.prolog` attribute. `query.source` holds the KB. The module `translate_rdf` exploits Thea2 to parse and translate RDF KBs into Prolog facts. It contains a modified version of the standard Thea2 library where we implemented some improvement for the management of annotation assertions to handle the probabilistic ones.

The `handleCreate` function is performed at the Pengine creation and was modified to allow the translation of the KB. The query given by the user is sent to the Pengine with the `ask` method, which executes the `parse` goal for the translation and then the query in this way:



```


<!-- http://cohse.semanticweb.org/ontologies/

<owl:NamedIndividual rdf:about="http://cohse.
    <rdf:type rdf:resource="http://cohse.sema
    <rdfs:label>Tom</rdfs:label>
    <rdfs:comment></rdfs:comment>
</owl:NamedIndividual>
<owl:Axiom>
    <disponte:probability>0.3</disponte:proba
    <owl:annotatedSource rdf:resource="http:/
    <owl:annotatedTarget rdf:resource="http:/
    <owl:annotatedProperty rdf:resource="&rdf
</owl:Axiom>




<!--
```

```
?- prob_instanceOf('natureLover','Kevin',Prob).
```

Figure 2. TRILL on SWISH ready to execute the query of Example 2. The left panel contains the KB while the bottom right panel a $prob\_instanceOf/3$ query.

```
function handleCreate() {
  var elem = this.pengine.options.runner;
  var data = elem.data(pluginName);
  ...
  this.pengine.ask("'$trill_on_swish wrapper'((" +
                   "parse,query_call(" +
                   termNoFullStop(data.query.query) +
                   ")))", options);
  elem.prologRunner('setState',"running");
  }
```

where `data.query.query` is a string containing the query and the predicate `query_call/1` expands if needed the arguments of the query and then calls it. The top right panel will then show the result of the query, see Figure 3.

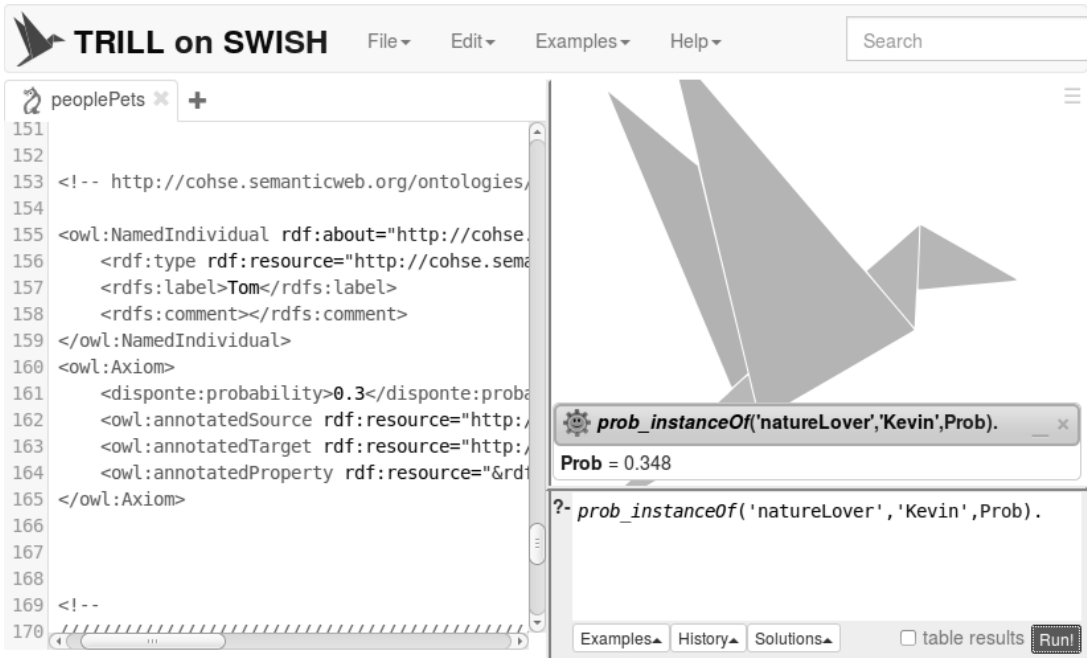
TRILL on SWISH can be opened with data preloaded in it. In order to preload a KB some parameters must be provided in the URL of TRILL on SWISH, in particular:

code The value can be either the text of the KB or a URL from which the KB is available and downloadable;

q The query to be set in the query panel.

For example the URL `http://trill.lamping.unife.it/trill_on_swish/?code=https://github.com/friguzzi/trill-on-swish/raw/master/examples/BRCA.owl` opens TRILL on SWISH preloading an owl file from GitHub.

We tested the robustness of the application by running two different stress tests. First we submitted queries without imposing a time limit for the executions. The queries and the KBs were chosen in order to saturate the main memory. Then, we set the time limit to 300 seconds, and we ran again all the queries. In both cases, the serves simply kills or interrupts the thread that exhaust the memory or that reaches the time limit without affecting the executions of other threads. An error message is returned to the client regarding the motivation for the execution interruption. It should be noted is that TRILL on SWISH is a testing tool useful for developing and experimenting also in



```
<!-- http://cohse.semanticweb.org/ontologies/

<owl:NamedIndividual rdf:about="http://cohse.
    <rdf:type rdf:resource="http://cohse.sema
    <rdfs:label>Tom</rdfs:label>
    <rdfs:comment></rdfs:comment>
</owl:NamedIndividual>
<owl:Axiom>
    <disponte:probability>0.3</disponte:proba
    <owl:annotatedSource rdf:resource="http:/
    <owl:annotatedTarget rdf:resource="http:/
    <owl:annotatedProperty rdf:resource="&rdf
</owl:Axiom>



<!--
```

```
prob_instanceOf('natureLover','Kevin',Prob).
Prob = 0.348
```

```
?- prob_instanceOf('natureLover','Kevin',Prob).
```

Figure 3. TRILL on SWISH after the execution of the query of Example 2. The resulting probability is shown in the top right panel.

a collaborative way, but it is not befitting heavy computations, for which a local installation should be used. For these reasons and to ensure the server responsiveness, we imposed a time limit on the query execution of 300 seconds. The tests show that the system is robust and can manage high loads even in case of errors in some threads.

6.1. Examples

TRILL on SWISH offers some example ontologies, available from the "Examples" menu, containing both probabilistic and certain axioms. In this section we show three examples.

The first ontology is the one of Example 2, which is shown in Figure 4 on page 16. The first two axioms are probabilistic, the former indicating that the individuals that own a pet are nature lovers with probability 0.5, the latter that a cat is a pet with probability 0.6. The remaining axioms are certain. The query can be written with the whole URL or in a short form, i.e. without the base part of the URL. By querying `prob_instanceOf('natureLover','Kevin',Prob).` we get `Prob=0.3`, as calculated in Example 2.

The second example is the BRCA ontology. This example could be seen as a use case of a real application of the proposed system. The axiom in Figure 5 on page 17 indicates that postmenopausal women who take estrogens can suffer a moderate increase of breast cancer risk with probability 0.67. If we want to compute the probability of a woman between 30 and 40 years old of getting breast tumor, we can express it as `sub_class('cancer_ra:WomanAged3040', 'WomanUnderLifetimeBRCRisk', Prob)` and obtain `Prob=0.123`. Note that in this example we use the namespace `cancer_ra` defined in the RDF/XML file.

The third ontology is a part of the Vicodi KB. Figure 6 on page 17 shows two probabilistic axioms. The former is a terminological axiom asserting that an artist is also a creator with probability equal to 0.85, the latter is an assertional axiom which means that Anthony van Dyck is a painter with probability 0.9. If we perform the query `prob_instanceOf('vicodi:Role','vicodi:Anthony-van-Dyck-is-Painter-in-Flanders', Prob),` TRILL returns

`Prob=0.2754`. This means that Anthony van Dyck had a role in European history with a 27.54% probability.

7. CONCLUSIONS

Web-based systems are, today, the way to reach out to a wider audience. In order to popularize the Probabilistic Semantic Web, we have developed the TRILL on SWISH web application that allows users to write and query probabilistic KBs following DISPONTE with just a web browser.

The program and the query are sent to the server, which returns the answer(s) to the user. TRILL on SWISH has been implemented by exploiting the features of the SWISH system for Prolog programming and querying on the Web and the `cplint` package for performing efficient inference by means of BDDs. TRILL on SWISH is available at `http://trill.lamping.unife.it` and already includes a number of examples that cover various domains, providing interesting applications of the Probabilistic Semantic Web.

A work analogous to TRILL on SWISH is cplint on SWISH [43]. As the name suggests, this application is based on SWISH and provides a web interface for the `cplint` suite.

In the months from July to November 2015 we monitored the accesses to `http://trill.lamping.unife.it` and we observed a total of 191 sessions with 83 different users and 404 page views, thus testifying the interest in the tool.

In the future, we are planning to include all the updates of SWISH in our system and to extend it with the possibility of expressing KBs with other OWL syntaxes (such as Functional, Turtle, etc.), and with learning algorithms. In addition we would like to integrate WebProtégé in TRILL on SWISH in order to allow users to build ontologies with a GUI. We are also planning to add the possibility to return the explanations in decreasing order with respect to their probability.

ACKNOWLEDGEMENT

We would like to thank Jan Wielemaker for his very useful comments on how to set up the system. This work was supported by the “GNCS-INdAM”.

```
...
<owl:Axiom>
  <disponte:probability rdf:datatype="&xsd;decimal">0.5</disponte:
      probability>
    <owl:annotatedTarget
      rdf:resource="http://cohse.semanticweb.org/ontologies/people#
          natureLover"/>
  <owl:annotatedProperty rdf:resource="&rdfs;subClassOf"/>
  <owl:annotatedSource>
    <owl:Restriction>
      <rdfs:subClassOf
        rdf:resource="http://cohse.semanticweb.org/ontologies/people#
            natureLover"/>
      <owl:onProperty
        rdf:resource="http://cohse.semanticweb.org/ontologies/people#
            has_animal"/>
      <owl:someValuesFrom
        rdf:resource="http://cohse.semanticweb.org/ontologies/people#
            pet"/>
    </owl:Restriction>
  </owl:annotatedSource>
</owl:Axiom>

<owl:Axiom>
  <disponte:probability rdf:datatype="&xsd;decimal">0.6</disponte:
      probability>
  <owl:annotatedSource
        rdf:resource="http://cohse.semanticweb.org/ontologies/people#
            cat"/>
  <owl:annotatedTarget
        rdf:resource="http://cohse.semanticweb.org/ontologies/people#
            pet"/>
  <owl:annotatedProperty
        rdf:resource="&rdfs;subClassOf"/>
</owl:Axiom>

<owl:NamedIndividual rdf:about="http://cohse.semanticweb.org/ontologies
    /people#Kevin">
  <rdfs:label>Kevin</rdfs:label>
  <has_animal
    rdf:resource="http://cohse.semanticweb.org/ontologies/people#Fluffy
        "/>
  <has_animal
    rdf:resource="http://cohse.semanticweb.org/ontologies/people#Tom"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://cohse.semanticweb.org/ontologies
    /people#Fluffy">
  <rdf:type rdf:resource="http://cohse.semanticweb.org/ontologies/
      people#cat"/>
  <rdfs:label>Fluffy</rdfs:label>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://cohse.semanticweb.org/ontologies
    /people#Tom">
  <rdf:type rdf:resource="http://cohse.semanticweb.org/ontologies/
      people#cat"/>
  <rdfs:label>Tom</rdfs:label>
</owl:NamedIndividual>
...
```

Figure 4. Axioms of the `people+pets` KB in RDF/XML syntax.

```
...
<rdf:RDF xmlns="&cancer_ra;"
     xml:base="http://clarkparsia.com/pronto/cancer_ra.owl#"
     xmlns:cancer_ra="http://clarkparsia.com/pronto/cancer_ra.owl#"
     ... >
...
<owl:Axiom>
  <disponte:probability
    rdf:datatype="&xsd;decimal">0.67</disponte:probability>
  <owl:annotatedSource
    rdf:resource="&cancer_ra;PostmenopausalWomanTakingEstrogen"/>
  <owl:annotatedTarget
    rdf:resource="&cancer_ra;WomanUnderModeratelyIncreasedBRCRisk"/>
  <owl:annotatedProperty rdf:resource="&rdfs;subClassOf"/>
</owl:Axiom>
...
```

Figure 5. An axiom and a snippet of the declaration of the namespaces of the BRCA KB expressed in RDF/XML syntax.

```
...
<owl:Axiom>
  <disponte:probability
    rdf:datatype="&xsd;decimal">0.85</disponte:probability>
  <owl:annotatedSource
    rdf:resource="http://vicodi.org/ontology#Artist"/>
  <owl:annotatedTarget rdf:resource="http://vicodi.org/ontology#Creator
      "/>
  <owl:annotatedProperty rdf:resource="&rdfs;subClassOf"/>
</owl:Axiom>
...
<owl:Axiom>
  <disponte:probability
    rdf:datatype="&xsd;decimal">0.9</disponte:probability>
  <owl:annotatedSource
    rdf:resource="http://vicodi.org/ontology#Anthony-van-Dyck-is-
        Painter-in-Flanders"/>
  <owl:annotatedTarget
    rdf:resource="http://vicodi.org/ontology#Painter"/>
  <owl:annotatedProperty rdf:resource="&rdf;type"/>
</owl:Axiom>
...
```

Figure 6. Axioms from the Vicodi KB in RDF/XML syntax.

REFERENCES

1. Hitzler P, Krötzsch M, Rudolph S. *Foundations of Semantic Web Technologies*. CRCPress, 2009.
2. Nilsson NJ. Probabilistic logic. *Artificial intelligence* 1986; **28**(1):71–87.
3. Halpern JY. An analysis of first-order logics of probability. *Artificial intelligence* 1990; **46**(3):311–350.
4. Bacchus F. *Representing and reasoning with probabilistic knowledge - a logical approach to probabilities*. MIT Press, 1990.
5. Sato T. A statistical learning method for logic programs with distribution semantics. *International Conference on Logic Programming*, MIT Press, Cambridge, MA: Tokyo, 1995; 715–729.
6. Poole D. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 1997; **94**(1-2):7–56.
7. Poole D. Probabilistic horn abduction and Bayesian networks. *Artificial intelligence* 1993; **64**(1):81–129.
8. Sato T, Kameya Y. PRISM: A language for symbolic-statistical modeling. *International Joint Conferences on Artificial Intelligence*, Morgan Kaufmann, Burlington, MA: Nagoya, Japan, 1997; 1330–1339.
9. Fuhr N. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science* 2000; **51**(2):95–110.
10. Vennekens J, Verbaeten S, Bruynooghe M. Logic programs with annotated disjunctions. *Logic Programming. International Conference on Logic Programming*, *Lecture Notes in Computer Science*, vol. 3131, Springer: Saint-Malo, France, 2004; 195–209.
11. Vennekens J, Denecker M, Bruynooghe M. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming* 2009; **9**(3):245–308.
12. De Raedt L, Kimmig A, Toivonen H. ProbLog: A probabilistic Prolog and its application in link discovery. *International Joint Conferences on Artificial Intelligence*, IJCAI/AAAI, Palo Alto, CA: Hyderabad, India, 2007; 2462–2467.
13. Bellodi E, Lamma E, Riguzzi F, Albani S. A distribution semantics for probabilistic ontologies. *International Workshop on Uncertainty Reasoning for the Semantic Web*, *CEUR Workshop Proceedings*, vol. 778, Sun SITE Central Europe: Bethlehem, Pennsylvania, USA, 2011.
14. Riguzzi F, Bellodi E, Lamma E. Probabilistic Datalog+/- under the distribution semantics. *International Workshop on Description Logics*, Kazakov Y, Lembo D, Wolter F (eds.), CEUR Workshop Proceedings, Sun SITE Central Europe: Rome, Italy, 2012.
15. Riguzzi F, Lamma E, Bellodi E, Zese R. Epistemic and statistical probabilistic ontologies. *International Workshop on Uncertainty Reasoning for the Semantic Web*, *CEUR Workshop Proceedings*, vol. 900, Sun SITE Central Europe: Boston, Massachusetts, USA, 2012; 3–14.
16. Riguzzi F, Bellodi E, Lamma E, Zese R. Probabilistic description logics under the distribution semantics. *Semantic Web-Interoperability, Usability, Applicability* 2015; **6**(5):477–501, doi:10.3233/SW-140154.
17. Sirin E, Parsia B, Cuenca-Grau B, Kalyanpur A, Katz Y. Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web* 2007; **5**(2):51–53.
18. Haarslev V, Hidde K, Möller R, Wessel M. The RacerPro knowledge representation and reasoning system. *Semantic Web* 2012; **3**(3):267–277.
19. Shearer R, Motik B, Horrocks I. Hermit: A highly-efficient owl reasoner. *International Workshop on OWL: Experiences and Directions*, *CEUR Workshop Proceedings*, vol. 432, Ruttenberg A, Sattler U, Dolbear C (eds.), Sun SITE Central Europe: Karlsruhe, Germany, 2008.
20. Klinov P. Pronto: A non-monotonic probabilistic description logic reasoner. *European Semantic Web Conference*, *Lecture Notes in Computer Science*, vol. 5021, Springer: Tenerife, Canary Islands, Spain, 2008; 822–826.
21. Zese R, Bellodi E, Lamma E, Riguzzi F, Aguiari F. Semantics and inference for probabilistic description logics. *Uncertainty Reasoning for the Semantic Web III*, *Lecture Notes in Computer Science*, vol. 8816. Springer, 2014; 79–99, doi:10.1007/978-3-319-13413-0_5. URL `http://ds.ing.unife.it/~friguzzi/Papers/RigBel14-URSWb-BC.pdf`.
22. Zese R, Bellodi E, Lamma E, Riguzzi F. A description logics tableau reasoner in Prolog. *Convegno Italiano di Logica Computazionale*, *CEUR Workshop Proceedings*, vol. 1068, Sun SITE Central Europe: Catania, Italy, 2013; 33–47. URL `http://ceur-ws.org/Vol-1068/paper-l02.pdf`.
23. Vassiliadis V, Wielemaker J, Mungall C. Processing OWL2 ontologies using Thea: An application of logic programming. *International Workshop on OWL: Experiences and Directions*, *CEUR Workshop Proceedings*, vol. 529, Sun SITE Central Europe: Chantilly, Virginia, USA, 2009.
24. Lager T, Wielemaker J. Pengines: Web logic programming made easy. *Theory and Practice of Logic Programming* 2014; **14**(4-5):539–552, doi:10.1017/S1471068414000192.
25. Schreiber G, Amin AK, Aroyo L, van Assem M, de Boer V, Hardman L, Hildebrand M, Omelayenko B, van Ossenbruggen J, Tordai A, *et al.*. Semantic annotation and search of cultural-heritage collections: The multimedian e-culture demonstrator. *Web Semantics: Science, Services and Agents on the World Wide Web* 2008; **6**(4):243–249, doi:10.1016/j.websem.2008.08.001. URL `http://dx.doi.org/10.1016/j.websem.2008.08.001`.
26. Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF ((eds.)). *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
27. Baader F, Horrocks I, Sattler U. Description logics. *Handbook of knowledge representation*. chap. 3, Elsevier, 2008; 135–179.
28. Patel-Schneider F P, Horrocks I, Bechhofer S. Tutorial on OWL 2003.
29. Riguzzi F, Bellodi E, Lamma E, Zese R. Probabilistic description logics under the distribution semantics. *Semantic Web - Interoperability, Usability, Applicability* 2015; **6**(5):447–501, doi:10.3233/SW-140154.
30. d'Amato C, Fanizzi N, Lukasiewicz T. Tractable reasoning with Bayesian description logics. *International Conference on Scalable Uncertainty Management*, *Lecture Notes in Computer Science*, vol. 5291, Springer, 2008; 146–159.

31. Ceylan II, Peñaloza R. Bayesian description logics. *Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014.*, *CEUR Workshop Proceedings*, vol. 1193, Bienvenu M, Ortiz M, Rosati R, Simkus M (eds.), CEUR-WS.org, 2014; 447–458.
32. Baader F, Peñaloza R. Axiom pinpointing in general tableaux. *Journal of Logic and Computation* 2010; **20**(1):5–34.
33. Kalyanpur A. Debugging and repair of OWL ontologies. PhD Thesis, The Graduate School of the University of Maryland 2006.
34. Baader F, Hollunder B. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning* 1995; **14**(1):149–180.
35. Valiant LG. The complexity of enumeration and reliability problems. *Society for Industrial and Applied Mathematics Journal on Computing* 1979; **8**(3):410–421.
36. Darwiche A, Marquis P. A knowledge compilation map. *Journal of Artificial Intelligence Research* 2002; **17**:229–264.
37. Meinel C, Slobodová A. *On the complexity of constructing optimal ordered binary decision diagrams*. Springer, 1994.
38. Santos Costa V, Rocha R, Damas L. The YAP Prolog system. *Theory and Practice of Logic Programming* 2012; **12**(1-2):5–34.
39. Wielemaker J, Schrijvers T, Triska M, Lager T. SWI-Prolog. *Theory and Practice of Logic Programming* 2012; **12**(1-2):67–96.
40. Wielemaker J, Huang Z, van der Meij L. SWI-Prolog and the web. *Theory and Practice of Logic Programming* 2008; **8**(3):363–392, doi:10.1017/S1471068407003237. URL `http://dx.doi.org/10.1017/S1471068407003237`.
41. Riguzzi F. Extended semantics and inference for the Independent Choice Logic. *Logic Journal of the IGPL* 2009; **17**(6):589–629, doi:10.1093/jigpal/jzp025.
42. Tudorache T, Nyulas C, Fridman Noy N, Musen MA. Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web* 2013; **4**(1):89–99, doi:10.3233/SW-2012-0057. URL `http://dx.doi.org/10.3233/SW-2012-0057`.
43. Riguzzi F, Bellodi E, Lamma E, Zese R, Cota G. Probabilistic logic programming on the web. *Software: Practice and Experience* 2015; doi:10.1002/spe.2386. URL `http://ds.ing.unife.it/~friguzzi/Papers/RigBelLam-SPE16.pdf`.