TRILL on SWISH Manual

May 12, 2017

1 Syntax

Description Logics (DLs) are knowledge representation formalisms that are at the basis of the Semantic Web [1, 2] and are used for modeling ontologies. They are represented using a syntax based on concepts, basically sets of individuals of the domain, and roles, sets of pairs of individuals of the domain. In this section, we recall the expressive description logic $\mathcal{ALC}$ [17]. We refer to [10] for a detailed description of $\mathcal{SHOIN}(\mathbf{D})$ DL, that is at the basis of OWL DL.

Let $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ be sets of *atomic concepts*, *roles* and *individuals*. A *role* is an atomic role $R \in \mathbf{R}$. *Concepts* are defined by induction as follows. Each $C \in \mathbf{A}$, $\bot$ and $\top$ are concepts. If C, C_1 and C_2 are concepts and $R \in \mathbf{R}$, then $(C_1 \sqcap C_2)$, $(C_1 \sqcup C_2)$, $\neg C$, $\exists R.C$, and $\forall R.C$ are concepts. Let C, D be concepts, $R \in \mathbf{R}$ and $a, b \in \mathbf{I}$. An *ABox* $\mathcal{A}$ is a finite set of *concept membership axioms* $a : C$ and *role membership axioms* $(a, b) : R$, while a *TBox* $\mathcal{T}$ is a finite set of *concept inclusion axioms* $C \sqsubseteq D$. $C \equiv D$ abbreviates $C \sqsubseteq D$ and $D \sqsubseteq C$.

A *knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A KB $\mathcal{K}$ is assigned a semantics in terms of set-theoretic 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 $R \in \mathbf{R}$.

TRILL allows the use of two different syntaxes used together or individually:

- RDF/XML
- TRILL syntax

RDF/XML syntax can be used by exploiting the predicate `owl_rdf/1`. For example:

```
owl_rdf('
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
```

```
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://here.the.IRI.of.your.ontology#"
     xml:base="http://here.the.IRI.of.your.ontology"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://here.the.IRI.of.your.ontology"/>

    <!--
    Axioms
    -->

</rdf:RDF>
').
```

For a brief introduction on RDF/XML syntax see *RDF/XML syntax and tools* section below (Sec. 1.2).

Note that each single `owl_rdf/1` must be self contained and well formatted, it must start and end with `rdf:RDF` tag and contain all necessary declarations (namespaces, entities, ...).

An example of the combination of both syntaxes is shown the example `johnEmployee.pl`. It models that *john* is an *employee* and that employees are *workers*, which are in turn people (modeled by the concept *person*).

```
owl_rdf('<?xml version="1.0"?>
<rdf:RDF xmlns="http://example.foo#"
     xml:base="http://example.foo"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://example.foo"/>

    <!-- Classes -->
    <owl:Class rdf:about="http://example.foo#worker">
        <rdfs:subClassOf rdf:resource="http://example.foo#person"/>
    </owl:Class>

</rdf:RDF>').

subClassOf('employee','worker').
```

```
owl_rdf('<?xml version="1.0"?>
<rdf:RDF xmlns="http://example.foo#"
     xml:base="http://example.foo"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://example.foo"/>

    <!-- Individuals -->
    <owl:NamedIndividual rdf:about="http://example.foo#john">
        <rdf:type rdf:resource="http://example.foo#employee"/>
    </owl:NamedIndividual>
</rdf:RDF>').
```

1.1 TRILL Syntax

1.1.1 Declarations

TRILL syntax allows, as in standard OWL, the declaration of classes, properties, etc.

```
class("classIRI").
datatype("datatypeIRI").
objectProperty("objectPropertyIRI").
dataProperty("dataPropertyIRI").
annotationProperty("annotationPropertyIRI").
namedIndividual("individualIRI").
```

However, TRILL properly works also in their absence.

TRILL syntax allows also the declaration of aliases for namespaces by using the `kb_prefix/2` predicate.

```
kb_prefix("foo","http://example.foo#").
```

After this declaration, the prefix `foo` is available, thus, instead of `http://example.foo#john`, one can write `foo:john`. It is possible to define also an empty prefix as

```
kb_prefix("","http://example.foo#").
```

or as

```
kb_prefix([],"http://example.foo#").
```

In this way `http://example.foo#john` can be written only as `john`.

Note: Only one prefix per alias is allowed. Aliases defined in OWL/RDF part have the precedence, in case more than one prefix was assigned to the same alias, TRILL keeps only the first assignment.

1.1.2 Axioms

Axioms are modeled using the following predicates

```
subClassOf("subClass","superClass").
equivalentClasses([list,of,classes]).
disjointClasses([list,of,classes]).
disjointUnion([list,of,classes]).

subPropertyOf("subPropertyIRI","superPropertyIRI").
equivalentProperties([list,of,properties,IRI]).
propertyDomain("propertyIRI","domainIRI").
propertyRange("propertyIRI","rangeIRI").
transitiveProperty("propertyIRI").
inverseProperties("propertyIRI","inversePropertyIRI").
symmetricProperty("propertyIRI").

sameIndividual([list,of,individuals]).
differentIndividuals([list,of,individuals]).

classAssertion("classIRI","individualIRI").
propertyAssertion("propertyIRI","subjectIRI","objectIRI").
annotationAssertion("annotationIRI",axiom,literal('value')).
```

For example, for asserting that *employee* is subclass of *worker* one can use

```
subClassOf(employee,worker).
```

while the assertion *worker* is equal to *workingman* can be defined as

```
equivalentClasses([worker,workingman]).
```

Annotation assertions can be defined, for example, as

```
annotationAssertion(foo:myAnnotation,
    subClassOf(employee,worker),'myValue').
```

In particular, an axiom can be annotated with a probability which defines the degree of belief in the truth of the axiom. See Section 2 for details.

Below, an example of an probabilistic axiom, following the TRILL syntax.

```
annotationAssertion('disponte:probability',
    subClassOf(employee,worker),literal('0.6')).
```

1.1.3 Concepts descriptions

Complex concepts can be defined using different operators.

Existential and universal quantifiers

```
someValuesFrom("propertyIRI","classIRI").
allValuesFrom("propertyIRI","classIRI").
```

Union and intersection of concepts

```
unionOf([list,of,classes]).
intersectionOf([list,of,classes]).
```

Cardinality descriptions

```
exactCardinality(cardinality,"propertyIRI").
exactCardinality(cardinality,"propertyIRI","classIRI").
maxCardinality(cardinality,"propertyIRI").
maxCardinality(cardinality,"propertyIRI","classIRI").
minCardinality(cardinality,"propertyIRI").
minCardinality(cardinality,"propertyIRI","classIRI").
```

Complement of a concept

```
complementOf("classIRI").
```

Nominal concept

```
oneOf([list,of,classes]).
```

For example, the class *workingman* is the intersection of *worker* with the union of *man* and *woman*. It can be defined as:

```
equivalentClasses([workingman,
    intersectionOf([worker,unionOf([man,woman])])]).
```

1.2 RDF/XML syntax and tools

As said before, TRILL is able to automatically translate RDF/XML knowledge bases when passed as a string using the preticate `owl_rdf/1`.

Consider the following axioms

```
classAssertion(Cat,fluffy)
subClassOf(Cat,Pet)
propertyAssertion(hasAnimal,kevin,fluffy)
```

The first axiom states that *fluffy* is a *Cat*. The second states that every *Cat* is also a *Pet*. The third states that the role *hasAnimal* links together *kevin* and *fluffy*.

RDF (Resource Descritpion Framework) is a standard W3C. See the syntax specification for more details. RDF is a standard XML-based used for representing knowledge by means of triples. A representations of the three axioms seen above is shown below.

```
<owl:NamedIndividual rdf:about="fluffy">
  <rdf:type rdf:resource="Cat"/>
</owl:NamedIndividual>

<owl:Class rdf:about="Cat">
  <rdfs:subClassOf rdf:resource="Pet"/>
</owl:Class>

<owl:ObjectProperty rdf:about="hasAnimal"/>
<owl:NamedIndividual rdf:about="kevin">
 <hasAnimal rdf:resource="fluffy"/>
</owl:NamedIndividual>
```

Annotations are assertable using an extension of RDF/XML. For example the annotated axiom below, defined using the TRILL sintax

```
annotationAssertion('disponte:probability',
    subClassOf('Cat','Pet'),literal('0.6')).
```

is modeled using RDF/XML syntax as

```
<owl:Class rdf:about="Cat">
 <rdfs:subClassOf rdf:resource="Pet"/>
</owl:Class>
<owl:Axiom>
 <disponte:probability rdf:datatype="&amp;xsd;decimal">
     0.6
 </disponte:probability>
 <owl:annotatedSource rdf:resource="Cat"/>
 <owl:annotatedTarget rdf:resource="Pet"/>
 <owl:annotatedProperty rdf:resource="&amp;rdfs;subClassOf"/>
</owl:Axiom>
```

If you define the annotated axiom in the RDF/XML part, the annotation must be declared in the knowledge base as follow

```
<!DOCTYPE rdf:RDF [
 ...
 <!ENTITY disponte "https://sites.google.com/a/unife.it/ml/disponte#" >
]>

<rdf:RDF
 ...
 xmlns:disponte="https://sites.google.com/a/unife.it/ml/disponte#"
 ...>

 ...
```

```
 <owl:AnnotationProperty rdf:about="&amp;disponte;probability"/>
 ...
</rdf:RDF>
```

There are many editors for developing knowledge bases.

2 Semantics

In the field of Probabilistic Logic Programming (PLP for short) many proposals have been presented. An effective and popular approach is the Distribution Semantics [14], which underlies many PLP languages such as PRISM [14, 15], Independent Choice Logic [12], Logic Programs with Annotated Disjunctions [19] and ProbLog [5]. Along this line, many reserchers proposed to combine probability theory with Description Logics (DLs for short) [10, 18]. DLs are at the basis of the Web Ontology Language (OWL for short), a family of knowledge representation formalisms used for modeling information of the Semantic Web

TRILL follows the DISPONTE [13, 20] semantics to compute the probability of queries. DISPONTE applies the distribution semantics [14] of probabilistic logic programming to DLs. A program following this semantics defines a probability distribution over normal logic programs called *worlds*. Then the distribution is extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs.

In DISPONTE, a *probabilistic knowledge base* $\mathcal{K}$ is a set of *certain axioms* or *probabilistic axioms* in which each axiom is independent evidence. Certain axioms take the form of regular DL axioms while probabilistic axioms are $p :: E$ where p is a real number in $[0, 1]$ and E is a DL axiom.

The idea of DISPONTE is to associate independent Boolean random variables to the probabilistic axioms. To obtain a *world*, we include every formula obtained from a certain axiom. For each probabilistic axiom, we decide whether to include it or not in w. A world therefore is a non probabilistic KB that can be assigned a semantics in the usual way. A query is entailed by a world if it is true in every model of the world.

The probability p can be interpreted as an *epistemic probability*, i.e., as the degree of our belief in axiom E. For example, a probabilistic concept membership axiom $p :: a : C$ means that we have degree of belief p in $C(a)$. A probabilistic concept inclusion axiom of the form $p :: C \sqsubseteq D$ represents our belief in the truth of $C \sqsubseteq D$ with probability p.

Formally, an *atomic choice* is a couple (E_i, k) where E_i is the ith probabilistic axiom and $k \in \{0, 1\}$. k indicates whether E_i is chosen to be included in a world ($k = 1$) or not ($k = 0$). A *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 formula). The probability of a composite choice κ is $P(\kappa) = \prod_{(E_i,1)\in\kappa} p_i \prod_{(E_i,0)\in\kappa}(1-p_i)$, where p_i is the probability associated with axiom E_i. A *selection* σ is a total composite choice, i.e., it contains an atomic choice (E_i, k) for every probabilistic axiom of the probabilistic KB. A selection σ identifies a theory w_σ called a *world* in this way:

$w_\sigma = \mathcal{C} \cup \{E_i | (E_i, 1) \in \sigma\}$ where $\mathcal{C}$ is the set of certain axioms. Let us indicate with $\mathcal{S}_\mathcal{K}$ the set of all selections and with $\mathcal{W}_\mathcal{K}$ the set of all worlds. The probability of a world w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(E_i,1)\in\sigma} p_i \prod_{(E_i,0)\in\sigma}(1 - p_i)$. $P(w_\sigma)$ is a probability distribution over worlds, i.e., $\sum_{w\in\mathcal{W}_\mathcal{K}} P(w) = 1$.

We can now assign probabilities to queries. Given a world w, the probability of a query Q is defined as $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of a query can be defined by marginalizing the joint probability of the query and the worlds, i.e. $P(Q) = \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q, w) = \sum_{w\in\mathcal{W}_\mathcal{K}} P(Q|w)p(w) = \sum_{w\in\mathcal{W}_\mathcal{K}:w\models Q} P(w)$.

Consider the following KB, inspired by the `people+pets` ontology [11]:

$$0.5 \;::\; \exists hasAnimal.Pet \sqsubseteq NatureLover \qquad 0.6 \;::\; Cat \sqsubseteq Pet$$
$$(kevin, tom) : hasAnimal \qquad (kevin, fluffy) : hasAnimal \qquad tom : Cat \qquad fluffy : Cat$$

The KB indicates that the individuals that own an animal which is a pet are nature lovers with a 50% probability and that *kevin* has the animals *fluffy* and *tom*. Fluffy and *tom* are cats and cats are pets with probability 60%. We associate a Boolean variable to each axiom as follow $F_1 = \exists hasAnimal.Pet \sqsubseteq NatureLover$, $F_2 = (kevin, fluffy) : hasAnimal$, $F_3 = (kevin, tom) : hasAnimal$, $F_4 = fluffy : Cat$, $F_5 = tom : Cat$ and $F_6 = Cat \sqsubseteq Pet$.

The KB has four worlds and the query axiom $Q = kevin : NatureLover$ is true in one of them, the one corresponding to the selection $\{(F_1, 1), (F_2, 1)\}$. The probability of the query is $P(Q) = 0.5 \cdot 0.6 = 0.3$.

Sometimes we have to combine knowledge from multiple, untrusted sources, each one with a different reliability. Consider a KB similar to the one of Example 2 but where we have a single cat, *fluffy*.

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

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

$$0.4 \;::\; fluffy : Cat \qquad 0.3 \;::\; fluffy : Cat$$

The two statements represent independent evidence on *fluffy* being a cat. We associate F_1 (F_2) to the first (second) probabilistic axiom.

The query axiom $Q = kevin : NatureLover$ is true in 3 out of the 4 worlds, those corresponding to the selections $\{\{(F_1, 1), (F_2, 1)\}, \{(F_1, 1), (F_2, 0)\}, \{(F_1, 0), (F_2, 1)\}\}$. So $P(Q) = 0.4 \cdot 0.3 + 0.4 \cdot 0.7 + 0.6 \cdot 0.3 = 0.58$. This is reasonable if the two sources can be considered as independent. In fact, the probability comes from the disjunction of two independent Boolean random variables with probabilities respectively 0.4 and 0.3: $P(Q) = P(X_1 \vee X_2) = P(X_1) + P(X_2) - P(X_1 \wedge X_2) = P(X_1) + P(X_2) - P(X_1)P(X_2) = 0.4 + 0.3 - 0.4 \cdot 0.3 = 0.58$

3 Inference

Traditionally, a reasoning algorithm decides whether an axiom is entailed or not by a KB by refutation: the axiom E is entailed if $\neg E$ has no model in the KB. Besides deciding whether an axiom is entailed by a KB, we want to find also explanations for the axiom, in order to compute the probability of the axiom.

3.1 Computing Queries Probability

The problem of finding explanations for a query has been investigated by various authors [16, 9, 8, 7, 6, 20]. It was called *axiom pinpointing* in [16] and considered as a non-standard reasoning service useful for tracing derivations and debugging ontologies. In particular, in [16] the authors define *minimal axiom sets* (*MinAs* for short). [MinA] Let $\mathcal{K}$ be a knowledge base and Q an axiom that follows from it, i.e., $\mathcal{K} \models Q$. We call a set $M \subseteq \mathcal{K}$ a *minimal axiom set* or *MinA* for Q in $\mathcal{K}$ if $M \models Q$ and it is minimal w.r.t. set inclusion. The problem of enumerating all MinAs is called MIN-A-ENUM. ALL-MINAS$(Q, \mathcal{K})$ is the set of all MinAs for query Q in knowledge base $\mathcal{K}$.

A *tableau* is a graph 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 to which the couple (a, b) belongs. Then, a set of consistency preserving tableau expansion rules are repeatedly applied 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 couple (C, a) where C and $\neg C$ are present in the label of a node, i.e. $C, \neg C \subseteq \mathcal{L}(a)$.

Some expansion rules are non-deterministic, i.e., they generate a finite set of tableaux. Thus the algorithm keeps a set of tableaux that is consistent if there is any tableau in it that is consistent, i.e., that is clash-free. Each time a clash is detected in a tableau G, the algorithm stops applying rules to G. Once every tableau in T contains a clash or no more expansion rules can be applied to it, the algorithm terminates. If all the tableaux in the final set T contain a clash, the algorithm returns unsatisfiable as no model can be found. Otherwise, any one clash-free completion graph in T represents a possible model for the concept and the algorithm returns satisfiable.

To compute the probability of a query, the explanations must be made mutually exclusive, so that the probability of each individual explanation is computed and summed with the others. To do that we assign independent Boolean random variables to the axioms contained in the explanations and defining the Disjunctive Normal Form (DNF) Boolean formula f_K which models the set of explanations. Thus $f_K(\mathbf{X}) = \bigvee_{\kappa \in K} \bigwedge_{(E_i,1)} X_i \bigwedge_{(E_i,0)} \overline{X_i}$ where $\mathbf{X} = \{X_i | (E_i, k) \in \kappa, \kappa \in K\}$ is the set of Boolean random variables. We can now translate f_K to a Binary Decision Diagram (BDD), from which we can compute the probability of the query with a dynamic programming algorithm that is linear in the size of the BDD.

In [3, 4] the authors consider the problem of finding a *pinpointing formula* instead of ALL-MINAS$(Q, \mathcal{K})$. The pinpointing formula is a monotone Boolean formula in which each Boolean variable corresponds to an axiom of the KB. This formula is built using the variables and the conjunction and disjunction connectives. It compactly encodes the set of all MinAs. Let's assume that each axiom E of a KB $\mathcal{K}$ is associated with a propositional variable, indicated with $var(E)$. The set of all propositional variables is indicated with $var(\mathcal{K})$. A valuation ν of a monotone Boolean formula is the set of propositional variables that are true. For a valuation $\nu \subseteq var(\mathcal{K})$, let $\mathcal{K}_\nu := \{t \in \mathcal{K} | var(t) \in \nu\}$. [Pinpointing formula] Given a query Q and a KB $\mathcal{K}$, a monotone Boolean formula ϕ over $var(\mathcal{K})$ is called a *pinpointing formula* for Q if for every valuation $\nu \subseteq var(\mathcal{K})$ it holds that $\mathcal{K}_\nu \models Q$ iff ν satisfies ϕ.

In Lemma 2.4 of [4] the authors proved that the set of all MinAs can be obtained by transforming the pinpointing formula into a Disjunctive Normal Form Boolean formula (DNF) and removing disjuncts implying other disjuncts.

Irrespective of which representation of the explanations we choose, a DNF or a general pinpointing formula, we can apply knowledge compilation and *transform it into a Binary Decision Diagram (BDD)*, from which we can compute the probability of the query with a dynamic programming algorithm that is linear in the size of the BDD.

We refer to [20, 21] for a detailed description of the two methods.

3.2 Possible Queries

TRILL can compute the probability or find an explanation of the following queries:

- Concept membership queries.
- Property assertion queries.
- Subsumption queries.
- Unsatifiability of a concept.
- Inconsistency of the knowledge base.

All the input arguments have to be atoms or ground terms. Note that it is necessary to specify which algorithm, TRILL or TRILL^P, has to be loaded for performing inference. This is done by using at the beginning of the input file the directive

```
:- trill.
```

for loading TRILL or

```
:- trillp.
```

for TRILL^P.

3.2.1 Probabilistic Queries

TRILL can be queried for computing the probability of queries. A resulting 0 probaility means that the query is false w.r.t. the knowledge base, while a probability value 1 that the query is certainly true.

The probability of an individual to belong to a concept can be asked using TRILL with the predicate

```
prob_instanceOf(+Concept:term,+Individual:atom,-Prob:double)
```

as in (`peoplePets.pl`)

```
?- prob_instanceOf(cat,'Tom',Prob).
```

The probability of two individuals to be related by a role can be computed with

```
prob_property_value(+Prop:atom,+Individual1:atom,
                    +Individual2:atom,-Prob:double)
```

as in (peoplePets.pl)

```
?- property_value(has_animal,'Kevin','Tom',Prob).
```

If you want to know tho probability with which a class is a subclass of another you have to use

```
prob_sub_class(+Concept:term,+SupConcept:term,-Prob:double)
```

as in (peoplePets.pl)

```
?- prob_sub_class(cat,pet,Prob).
```

The probability of the unsatisfiability of a concept can be asked with the predicate

```
prob_unsat(+Concept:term,-Prob:double)
```

as in (peoplePets.pl)

```
?- prob_unsat(intersectionOf([cat,complementOf(pet)]),P).
```

This query for example corresponds with a subsumption query, which is represented as the intersection of the subclass and the complement of the superclass.

Finally, you can ask the probability if the inconsistency of the knowledge base with

```
prob_inconsistent_theory(-Prob:double)
```

3.2.2 Non Probabilistic Queries

In TRILL you can also ask whether a query is true or false w.r.t. the knowledge base and in case of a succesful query an explanation can be returned as well. Query predicates in this case differs in the number of arguments, in the second case, when we want also an explanation, an extra argument is added to unify with the list of axioms build to explain the query.

The query if an individual belongs to a concept can be used the predicates

```
instanceOf(+Concept:term,+Individual:atom)
instanceOf(+Concept:term,+Individual:atom,-Expl:list)
```

as in (peoplePets.pl)

```
?- instanceOf(pet,'Tom').
?- instanceOf(pet,'Tom',Expl).
```

In the first query the result is `true` because Tom belongs to cat, in the second case TRILL returns the explanation

```
[classAssertion(cat,'Tom'), subClassOf(cat,pet)]
```

Similarly, to ask whether two individuals are related by a role you have to use the queries

```
property_value(+Prop:atom,+Individual1:atom,+Individual2:atom)
property_value(+Prop:atom,+Individual1:atom,
               +Individual2:atom,-Expl:list)
```

as in (peoplePets.pl)

```
?- property_value(has_animal,'Kevin','Tom').
?- property_value(has_animal,'Kevin','Tom',Expl).
```

If you want to know if a class is a subclass of another you have to use

```
sub_class(+Concept:term,+SupConcept:term)
sub_class(+Concept:term,+SupConcept:term,-Expl:list)
```

as in (peoplePets.pl)

```
?- sub_class(cat,pet).
?- sub_class(cat,pet,Expl).
```

The unsatisfiability of a concept can be asked with the predicate

```
unsat(+Concept:term)
unsat(+Concept:term,-Expl:list)
```

as in (peoplePets.pl)

```
?- unsat(intersectionOf([cat,complementOf(pet)])).
?- unsat(intersectionOf([cat,complementOf(pet)]),Expl).
```

In this case, the returned explanation is the same obtained by querying if cat is subclass of pet with the `sub_class/3` predicate, i.e., `[subClassOf(cat,pet)]`

Finally, you can ask about the inconsistency of the knowledge base with

```
inconsistent_theory
inconsistent_theory(-Expl:list)
```

3.3 TRILL Useful Predicates

There are other predicates defined in TRILL which helps manage and load the KB.

```
add_kb_prefix(++ShortPref:string,++LongPref:string)
add_kb_prefixes(++Prefixes:list)
```

They register the alias for prefixes. The firs registers `ShortPref` for the prefix `LongPref`, while the second register all the alias prefixes contained in Prefixes. The input list must contain pairs alias=prefix, i.e., `[('foo'='http://example.foo#')]`. In both cases, the empty string `''` can be defined as alias. The predicates

```
remove_kb_prefix(++ShortPref:string,++LongPref:string)
remove_kb_prefix(++Name:string)
```

remove from the registered aliases the one given in input. In particular, `remove_kb_prefix/1` takes as input a string that can be an alias or a prefix and removes the pair containing the string from the registered aliases.

```
add_axiom(++Axiom:axiom)
add_axioms(++Axioms:list)
```

These predicates add (all) the given axiom to the knowledge base. While, to remove axioms can be similarly used the predicates

```
remove_axiom(++Axiom:axiom)
remove_axioms(++Axioms:list)
```

All the axioms must be defined following the TRILL syntax.

Finally, we can interrogate TRILL to return the loaded axioms with

```
axiom(?Axiom:axiom)
```

This predicate searches in the loaded knowledge base axioms that unify with Axiom.

4 Download Query Results through an API

The results of queries can also be downloaded programmatically by directly approaching the Pengine API. Example client code is available. For example, the `swish-ask.sh` client can be used with bash to download the results for a query in CSV. The call below downloads a CSV file for the coin example.

```
$ bash swish-ask.sh --server=http://trill.lamping.unife.it \
  examples/trill/peoplePets.pl \
  Prob "prob_instanceOf('natureLover','Kevin',Prob)"
```

The script can ask queries against Prolog scripts stored in `http://trill.lamping.unife.it` by specifying the script on the commandline. User defined files stored in TRILL on SWISH (locations of type `http://trill.lamping.unife.it/p/johnEmployee_user.pl`) can be directly used, for example:

```
$ bash swish-ask.sh --server=http://trill.lamping.unife.it \
  johnEmployee_user.pl Expl "instanceOf(person,john,Expl)"
```

Example programs can be used by specifying the folder portion of the url of the example, as in the first johnEmployee example above where the url for the program is `http://trill.lamping.unife.it/examples/trill/johnEmployee.pl`.

You can also use an url for the program as in

```
$ bash swish-ask.sh --server=http://trill.lamping.unife.it \
  https://raw.githubusercontent.com/friguzzi/trill-on-swish/\
  master/examples/trill/peoplePets.pl \
  Prob "prob_instanceOf('natureLover','Kevin',Prob)"
```

Results can be downloaded in JSON using the option `--json-s` or `--json-html`. With the first the output is in a simple string format where Prolog terms are sent using quoted write, the latter serialize responses as HTML strings. E.g.

```
$ bash swish-ask.sh --json-s --server=http://trill.lamping.unife.it \
  johnEmployee_user.pl Expl "instanceOf(person,john,Expl)"
```

The JSON format can also be modified. See `http://www.swi-prolog.org/pldoc/doc_for?object=pengines%3Aevent_to_json/4`.

Prolog can exploit the Pengine API directly. For example, the above can be called as:

```
?- [library(pengines)].
?- pengine_rpc('http://trill.lamping.unife.it',
     prob_instanceOf('natureLover','Kevin',Prob),
     [ src_url('https://raw.githubusercontent.com/friguzzi/trill-on-swish/\
  master/examples/trill/peoplePets.pl'),
     application(swish)
     ]).
Prob = 0.51.
?-
```

5 Manual in PDF

A PDF version of the manual is available at `https://github.com/rzese/trill/blob/master/doc/help-trill.pdf`.

6 Bibliography

References

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

[2] F. Baader, I. Horrocks, and U. Sattler. Description logics. In *Handbook of knowledge representation*, chapter 3, pages 135–179. Elsevier, 2008.

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

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

[5] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, pages 2462–2467, 2007.

[6] C. Halaschek-Wiener, A. Kalyanpur, and B. Parsia. Extending tableau tracing for ABox updates. Technical report, University of Maryland, 2006.

[7] A. Kalyanpur. *Debugging and Repair of OWL Ontologies.* PhD thesis, The Graduate School of the University of Maryland, 2006.

[8] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *ISWC*, volume 4825 of *LNCS*, pages 267–280. Springer, 2007.

[9] A. Kalyanpur, B. Parsia, E. Sirin, and J. A. Hendler. Debugging unsatisfiable classes in OWL ontologies. *J. Web Sem.*, 3(4):268–293, 2005.

[10] T. Lukasiewicz and U. Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *J. Web Sem.*, 6(4):291–308, 2008.

[11] F. Patel-Schneider, P, I. Horrocks, and S. Bechhofer. Tutorial on OWL, 2003.

[12] D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artif. Intell.*, 94(1-2):7–56, 1997.

[13] Fabrizio Riguzzi, Evelina Lamma, Elena Bellodi, and Riccardo Zese. Epistemic and statistical probabilistic ontologies. In *URSW*, volume 900 of *CEUR Workshop Proceedings*, pages 3–14. Sun SITE Central Europe, 2012.

[14] T. Sato. A statistical learning method for logic programs with distribution semantics. In *ICLP*, pages 715–729. MIT Press, 1995.

[15] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res.*, 15:391–454, 2001.

[16] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *IJCAI*, pages 355–362. Morgan Kaufmann, 2003.

[17] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[18] Umberto Straccia. Managing uncertainty and vagueness in description logics, logic programs and description logic programs. In *International Summer School on Reasoning Web*, volume 5224 of *LNCS*, pages 54–103. Springer, 2008.

[19] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *ICLP*, volume 3131 of *LNCS*, pages 195–209. Springer, 2004.

[20] Riccardo Zese. *Probabilistic Semantic Web*, volume 28 of *Studies on the Semantic Web*. IOS Press, 2017.

[21] Riccardo Zese, Elena Bellodi, Fabrizio Riguzzi, Giuseppe Cota, and Evelina Lamma. Tableau reasoning for description logics and its extension to probabilities. *Ann. Math. Artif. Intel.*, pages 1–30, 2016.