A Meta-logical Approach for Reasoning with Ontologies and Rules Expressed In OWL 2

VISIT HIRANKITTI and TRANG MAI XUAN
Department of Computer Engineering
King Mongkut’s Institute of Technology Ladkrabang
Ladkrabang, Bangkok 10520, Thailand
THAILAND
v_hirankitti@yahoo.com and trangmx@gmail.com

Abstract: - OWL is a decidable language for representing ontologies, it adds considerable expressive power to the Semantic Web. However, it has several shortcomings in relational expressivity. These limitations can be overcome by extending an OWL ontology with some form of rules. The revision 2 of OWL is much richer than its predecessor with respect to modeling with rules. In this paper we propose a framework for reasoning with ontology and rule by adopting meta-logic. Our meta-logical system consists of several meta-programs expressing rules and ontologies at meta-level, and an inference engine in form of a meta-interpreter defined by a demo(.) predicate. Our meta-reasoning framework can reason with ontologies and rules expressed in OWL 2 by: (1) mapping statements expressed in ontology to meta-statements, and mapping rules to meta-rules, (2) the meta-statements and meta-rules, then, are reasoned by a meta-interpreter which provides a query answering mechanism to infer meta and context information from the Semantic Web Ontology.

Key-Words: - Description Logics, Metalogic Programming, OWL 2, Rules, Semantic Web Ontologies.

1 Introduction

Ontologies and rules play an important role in Semantic Web (or shortly ‘SW’). Ontology forms a vocabulary and axioms that can be used to express knowledge base and that can be used for sharing knowledge on the web. The W3C view of ontologies is reflected by the Web Ontology Language OWL. Its core version OWL-DL is essentially an XML encoding of an expressive Description Logic (DL), builds upon RDF (Resource Description Framework) and includes a substantial fragment of RDF-Schema (RDFS). The vocabularies defined in such ontologies consist of classes and properties, called also, respectively, concepts and roles. They also have well-defined logical semantics, where they are considered as, respectively unary and binary predicates, and interpreted as relations. OWL has been successfully applied to many problems in computer science, and it has rapidly become a de facto standard for ontology development and data interchange. However, some knowledge can be formulated more natural in terms of rules than in terms of other kinds of ontological axioms.

Ruled-based formalism is another popular paradigm of knowledge representation grounded in logic programming. It has been proposed as a possible solution for modeling knowledge which complements the means of specifying knowledge expressed by OWL. In broadest sense, a rule could be any statement of the form “if the precondition $p$ holds then the conclusion $c$ holds”, where the precondition and the conclusion are logical properties. The realization of the rule will allow further means to deduce knowledge and combine information. This lead to a way for enhancing content of ontology and reasoning capabilities is to integrate rules with ontology languages, such as OWL.

The integration of ontologies and rules has recently considerable attention in research on ontologies and Semantic Web, and many proposals in this direction have been made. One of the ideas of the integration approaches is to combine DL with first-order Horn-logic rule. This is the basis of the Semantic Web Rule Language SWRL [2], proposed as a rule extension to OWL. However, SWRL becomes undecidable even when the rules are function-free [2]. In order to make it decidable, more restricted rule languages have been taken into account, such as DL-safe rules [7] or Description Logic Programs DLP [1]. Recently, the new revision 2 of OWL (called OWL 2) has been developed by W3C. OWL 2 is much richer than its predecessor OWL with respect to modeling with rules, it based on DL SROIQ [11] which can completely internalize DL rules as decidable fragment of SWRL.

In previous work [9], we have developed a meta-logical approach for reasoning with semantic web ontologies expressed in RDF, RDFS, and OWL. In this paper we enhance the framework so that it can reason with integration of ontologies and rules in OWL 2.

The remainder of the paper is organized as follows. Section 2 reviews concepts of integrating ontologies and
rules, and shows how rules can be expressed in OWL 2. Accordingly, we extend our previous work to reason with ontologies and rules expressed in OWL 2 in Section 3. Section 4 demonstrates how our framework reasons with ontologies and rules. Finally, section 5 concludes this paper.

2 Integrating Ontologies and Rules

2.1 The Concepts

The issue of having rules on top or aside ontologies written in OWL is an important milestone in the research on Semantic Web for completing the Semantic Web architecture, and many proposals in this direction have been made. They range from hybrid approaches to homogeneous approaches. The idea of former approaches is that, the predicates of the rules and the ontologies are distinguished and suitable interfacing between them is facilitated. There are some research works on these approaches, such as, ALC-log language, a hybrid integration of Datalog and Description Logic ALC, has been described in [4], CARIN, another hybrid integration of Datalog with different DLs, was defined in [5], and in [6] a hybrid integration of OWL DL (or more precisely the DL SHOIN) with normal rules under answer set semantics has been presented. In the latter approaches both rules and ontologies are combined in the same logical language without making a priori distinction between the rule predicates and the ontology predicates. Two examples of these approaches are Description Logic Program (DLP) [1] and Semantic Web Rule Language (SWRL) [2]. In [1] a DLP was proposed by combining Description Logics (DLs) which is the basis for the Ontology languages and Logic Programs which is the basis for rule languages. It supports a bidirectional translation of premises and inferences from the fragment of DL to LP, and vice versa from the fragment of LP to DL. This translation enables one to build rules on top of ontologies. SWRL was presented in [2] as a new language for integration of rules and ontologies, in which OWL was extended with rule in form of Horn clause rules expressed in RuleML.

SWRL becomes the most commonly used ones on describing integration of rules and ontologies. However, the straightforward addition of the rules to ontologies is main cause of undecidability for reasoning in SWRL. In order to retain decidability some restrictions have been added to SWRL rules, and then such kind of rules can be rewritten into a set of DL axioms using the features introduced in SHOIN. This technique has been presented in [8]. Due to the fact that OWL 2 has been developed based on SHOIN Description Logic, OWL 2 can express rules in form of DL axioms.

2.2 Express Rules in OWL 2

The previous SW ontology languages such as DAML+OIL and OWL are developed based on SHOIN DL [12]. SHOIN provides a variety of constructors for building class expressions. The DL class expressions can be seen in terms of a correspondence to first order logic (FOL) which is basis for rules. Table 1 shows the FOL formulae corresponding to the DL class expressions.

<table>
<thead>
<tr>
<th>Expression</th>
<th>DL</th>
<th>FOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>subclassOf</td>
<td>C ⊑ D</td>
<td>∀ x. C(x) → D(x)</td>
</tr>
<tr>
<td>subpropertyOf</td>
<td>P₁ ⊑ P₂</td>
<td>∀ x,y,P₁(x,y) → P₂(x,y)</td>
</tr>
<tr>
<td>transitiveProperty</td>
<td>P* ⊑ P</td>
<td>∀ x,y,z,(P(x,y) ∨ P(y,z)) → P(x,z)</td>
</tr>
<tr>
<td>functionalProperty</td>
<td>T ≤ 1 P</td>
<td>∀ x,y,z,(P(x,y) ∨ P(z,z)) → y=z</td>
</tr>
<tr>
<td>inverseProperty</td>
<td>P = Q</td>
<td>∀ x,y,P(x,y) → Q(y,x)</td>
</tr>
<tr>
<td>intersectionOf</td>
<td>C₁ ∩ ... ∩ Cₙ</td>
<td>C₁(x) ∧ ... ∧ Cₙ(x)</td>
</tr>
<tr>
<td>unionOf</td>
<td>C₁ ∪ ... ∪ Cₙ</td>
<td>C₁(x) ∨ ... ∨ Cₙ(x)</td>
</tr>
<tr>
<td>complementOf</td>
<td>¬C</td>
<td>¬C(x)</td>
</tr>
<tr>
<td>oneOf</td>
<td>{a₁,...,a_n}</td>
<td>x=a₁ ∨ ... ∨ x=a_n</td>
</tr>
<tr>
<td>hasClass</td>
<td>⊔PC</td>
<td>∀ y,P(x,y) ∧ C(y)</td>
</tr>
<tr>
<td>toClass</td>
<td>∀ PC</td>
<td>∀ y,P(x,y) → C(y)</td>
</tr>
</tbody>
</table>

Table 1: DL FOL equivalence

Using such formulae, some FOL can be written in DL as well as in DAML+OIL or OWL. For example: A rule has form: C(x) ∧ ¬D(x) → E(x) ∨ F(x) can be rewritten to DL axiom like: C ⊓ ¬D ⊑ E ∪ F. A rule: C(x) ∧ R(x,y) → E(x) can be encoded in DL like: C ⊓ R ⊑ E. However, with some rules such as:

(a) hasParent(x,y) ∧ hasBrother(y,z) → hasUncle(x,z),
(b) hasParent(x,y) ∧ hasChild(x,y) → fatherOf(x,y).

There are no above equivalence can be used to rewrite these rules into DL axioms. OWL 2, based on DL SROIQ, supports Role Inclusion Axiom (RIA), and Self concept to describe more forms of rule. Role Inclusion Axioms are constructs of the form R o S ⊑ T where o is a binary composition operator. This form is equivalent with FOL formulae: ∃ x,y,z.(R(x,y) ∧ S(y,z)) → T(x,z). By using this feature, rule (a) can be easy rewritten in DL axioms as:

hasParent o hasBrother ⊑ hasUncle.

Self concept allows to express “local reflexivity”, i.e., to model a role that has the same individual for antecedent and successor. Self can be used to transform classes to properties, i.e., whenever we encounter an role of the form R(x,x), we can introduce a new class Cᵣ which we define with an axiom Cᵣ |= ∃R.Self, and vice versa. By using this feature, we can describe rule (b) in
DL axioms. Firstly, we transform class $\text{Man}$ to property $P_{\text{Man}}$ by define new axiom $\text{Man} \equiv 3P_{\text{Man}} \cdot \text{Self}$, then we use RIA. Rule (b) corresponds to DL axioms as follow:

$\text{Man} \equiv 3P_{\text{Man}} \cdot \text{Self}$.

$P_{\text{Man}} \circ \text{hasChild} \sqsubseteq \text{fatherOf}$.

With the new features, we can express a quite wide range of rules, if they satisfy certain restrictions [8], in form of OWL 2 axioms. In next sections we enhance our previous framework [9] so that it can reason with rules and ontologies expressed in OWL 2.

3 Our Meta-logical Approach

3.1 Our Approach

Our framework forms a logical system consisting of meta-programs and an inference engine. The former is in the form of logical sentences representing a meta-level description of an SW ontology. That is, the ontology described by OWL 2 is transformed into a meta-logical representation. The later is a meta-interpreter, in the form of a demo (meta-)program, which is used to infer explicit and implicit information, or in other words draw conclusions, from the former. The meta-interpreter also communicate to the Internet to obtain SW ontologies, communicate with the user to get SW information, draw inference consequences for the user, and traverse a link to an SW or web resource like a web browser. Our framework presented in this paper is enhanced with the purpose to facilitate the system to reason with ontologies and rules expressed in OWL 2. Our meta-logical system can be simply illustrated as in Fig. 1. In order to deal with rules, expressed by using new features of OWL 2, the meta-program in our framework has to add new statements corresponding with rules.

To explain our framework, in the next three subsections we first introduce our meta-language used for formulating the meta-programs of ontologies and rules, then explain the meta-programs in details. Finally we describe our meta-interpreter.

3.2 Meta-language for an SW Ontology

The language elements of an SW ontology are classes, properties, instances, and relationships between/among them described in the object level and the meta-level as depicted in Fig. 2. At the object level, an instance can be an individual or a literal of a domain, e.g. ‘john’, and property is a relationship between individuals, or is an individual’s attribute, e.g. ‘hasSon’, ‘type’. At the meta-level, a meta-instance can be an individual, a property, a class, or an object-level statement. A meta-property is a property to describe a meta-instance’s attribute or a relationship between/among meta-instances, e.g. ‘reflexive’, etc. Notice that according to the SW convention, to make a name appearing in an ontology unique, we qualify it with a namespace like $<\text{namespace}>:<\text{name}>$, such as ‘f’:‘son’, ‘f’:‘hasSon’, ‘owl’:‘reflexive’, etc. Henceforth this qualified name will be used throughout.

According to our framework, in an SW ontology we distinguish between its object and meta levels, and similarly its object and meta languages. The object language specifies objects and their relationships in the real world. The meta-language describes the syntactic form of the object language. Hence, we have formulated two meta-languages: one discussing mainly about objects and their relationships we call it “meta-language for the object level (ML)”, and the other we call “meta-language for the meta-level (MML)”, which discusses mainly about classes, instances, properties and their relationships.

- **Meta-language for the object level (ML)**
  Objects and their relationships at the object level are specified in an SW ontology and this information is expressed by the elements of ML below.

- **Meta-constant** specifies a name of an object and a literal, e.g. ‘son’, including a reference, e.g. a namespace, the latter is a meta-constant of MML. This means that ML and MML are not totally separated.

- **Meta-variable** stands for a different meta-constant at a different time, e.g. Person.

- **Meta-function symbol** stands for a name of a relation between objects, or a name of an object’s property—i.e. an object-level predicate name, such as ‘hasSon’, ‘name’. It also stands for other meta-level function symbol, e.g. ‘‘:’’, ‘‘:, ‘‘type’’.

- **Meta-term** is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-
terms, e.g. ‘f’ : ‘hasSon’, ‘owl’ : ‘reflexive’. To express object-level predicate it has the form: P(S, O), where P is an object-level predicate name, S and O are meta-constants or meta-variable, e.g. ‘f’ : ‘hasSon’ (‘f’ : ‘fa’, ‘f’ : ‘son’).

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has the form: statement(object-level-sentence), e.g.

\[
\text{statement(‘f’ : ‘hasSon’ (‘f’ : ‘fa’, ‘f’ : ‘son’)) \leftrightarrow true) .}
\]

- **Meta-language for the meta-level (MML)**

Apart from the object language, an SW ontology also defines classes, properties, their relationships, as well as class-instance relations, and we argue that this information is meta-information of the object level. Here we express this information by MML which includes:

- **Meta-constant** specifying a name of an instance, a property, a class, a literal, and a namespace.

- **Meta-variable** standing for a different meta-constant at a different time.

- **Meta-function symbol** standing for a logical connective, e.g. ‘\(\leftarrow\)’, ‘\(\land\)’; or ‘\(\:\)’; or a name of set operators applied on classes such as union; or a meta-predicate name being a name of a relation between entities; or a name of characteristic of a property, which may fall into one of the following categories:
  - Class-class relations: equivalent class of, etc.
  - Class-instance relations: instance of, class of, etc.
  - Property-property relations: property chain of, etc.
  - Class-property relations: Keys, etc.

**Relations between literals and instances/classes/properties:** we can take these relations as attributes of instances, of classes, or of properties, e.g. comment.

**Characteristics of properties:** reflexive, asymmetric, etc.

- **Meta-term** being either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of meta-terms, e.g. ‘f’ : ‘fatherOf’, etc. When a meta-term expresses a meta-level predicate stating a relation between entities, it has the form of \(\text{Pred(Sub, Obj)}\), and when it expresses a meta-level predicate stating a characteristic of a property, it has the form of \(\text{Pred(Prop)}\), where \(\text{Pred}\) is a meta-predicate name, \(\text{Sub, Obj}\), and \(\text{Prop}\) (a property) are meta-constants or meta-variables.

The meta-term expressing a meta-level sentence is a term \(\text{Pred(Sub, Obj)}\) or \(\text{Pred(Prop)}\) or a logical-connector function symbol applied to the tuple of these terms. Let all meta-variables appearing in the meta-level sentence be universally quantified. One form of the sentence is a Horn-clause meta-rule, e.g.

‘owl’ : ‘propertyDisjointWith’ (P, DP) \(\leftrightarrow\) ‘owl’ : ‘propertyDisjointWith’ (DP, P).

- **Meta-statement** being a meta-predicate or meta-predicates connected by logical connective. It has two forms \(\text{meta-statement(meta-level-sentence)}\) and \(\text{axiom(meta-level-sentence)}\), the latter represents a rule for a mathematical axiom, e.g.:

\[
\text{meta-statement(‘owl’ : ‘propertyDisjointWith’ (‘f’ : ‘likes’, ‘f’ : ‘dislikes’)) \leftrightarrow true).}
\]

\[
\text{axiom(‘owl’ : ‘propertyDisjointWith’ (P, DP)) \leftrightarrow ‘owl’ : ‘propertyDisjointWith’ (DP, P).}
\]

### 3.3 Meta-programs of Ontology and Rule

Each SW ontology is transformed into a meta-program containing a (sub-)meta-program expressed in ML, called MP, and a (sub-)meta-program expressed in MML, called MMP. Another meta-program expresses some mathematical axioms for classes and properties called AMP is also needed for the inference engine to reason with MP and MMP.

- **Meta-program for the object level (MP)**

MP contains information of instances and their relationship in forms of meta-statements for the object level: \(\text{statement(P(S,O) \leftrightarrow true)}\). In term of the rule system, it can be understood as facts. Here is an example of MP:

\[
\text{statement(‘f’ : ‘hasFather’ (‘f’ : ‘M02’, ‘f’ : ‘M01’) \leftrightarrow true).}
\]

\[
\text{statement(‘f’ : ‘hasSibling’ (‘f’ : ‘M01’, ‘f’ : ‘F01’) \leftrightarrow true).}
\]

- **Meta-program for the meta-level (MMP)**

MMP contains meta-statements for classes, properties, their relationships, and class-instance relationships in terms of meta-rules. Here are some typical examples:

//meta-statement about classes and their relationships
\[
\text{meta-statement(‘rdfs’ : ‘subClassOf’ (C, SC) \leftrightarrow true).}
\]

\[
\text{meta-statement(‘rdfs’ : ‘equivalentClass’ (C, EC) \leftrightarrow true).}
\]

\[
\text{meta-statement(‘owl’ : ‘domain’ (P, D) \leftrightarrow true).}
\]

\[
\text{meta-statement(‘owl’ : ‘range’ (P, R) \leftrightarrow true).}
\]

\[
\text{meta-statement(‘rdfs’ : ‘type’ (I, C) \leftrightarrow true).}
\]

...
3.4 The Meta-interpreter

The meta-interpreter in our framework is built to reason with the meta-programs MPs, MMPs, and AMPs and can be used to develop an intelligent agent to reason with SW ontologies. It is defined by a demo predicate of the form demo(A). With this predicate we can infer the answer A from the meta-programs. Our meta-interpreter adapts the Vanilla meta-interpreter in [10] in order for reasoning with the meta-programs transformed from ontologies and rules where we have defined some kinds of meta-level statements: (1) statement(A ← B) for the object-level of an ontology, (2) meta_statement(A ← B) for the meta-level of an ontology, and (3) axiom(A ← B) for the mathematical axioms. The definition of demo/1 is:

\[
\text{demo}(true).
\]

The first clause (true) is the basic case for proving an atom to be true. The second clause (conj) is used for proving a conjunction goal. Three last clauses (ost), (mst), and (ast) are used for interpreting three meta-statements of the three meta-programs MP, MMP, and AMP respectively.

4 Query Answering with Our Framework

In our framework ontologies and rules expressed in OWL 2 are transformed into meta-programs. The meta-programs are formed by three sub-meta-programs MP, MMP and AMP. The meta-programs are used as inputs of meta-interpreter which is implemented in Prolog. The meta-interpreter is an inference engine reasoning with meta-programs to derive conclusions from SW ontologies and rules.

The family ontology from [13] is an example to test new features of OWL 2. We add some rules to this ontology by using OWL 2. After it is transformed into meta-programs, here we show a part of them:

- **The MP program:**

  \[
  \text{statement('f': 'hasParent')
  \begin{array}{l}
  ('f': 'M02', 'f': 'M01') \leftrightarrow \text{true}.
  \end{array}
  \]

  \[
  \text{statement('f': 'hasParent')
  \begin{array}{l}
  ('f': 'F02', 'f': 'M01') \leftrightarrow \text{true}.
  \end{array}
  \]

  \[
  \text{statement('f': 'hasParent')
  \begin{array}{l}
  ('f': 'M03', 'f': 'F02') \leftrightarrow \text{true}.
  \end{array}
  \]

- **The MMP program:**

  \[
  \text{meta_statement('rdf': 'type')
  \begin{array}{l}
  ('f': 'M01', 'f': 'Man') \leftrightarrow \text{true}.
  \end{array}
  \]

  \[
  \text{meta_statement('rdf': 'type')
  \begin{array}{l}
  ('f': 'M02', 'f': 'Man') \leftrightarrow \text{true}.
  \end{array}
  \]

  \[
  \text{meta_statement('rdf': 'type')
  \begin{array}{l}
  ('f': 'M03', 'f': 'WoMan') \leftrightarrow \text{true}.
  \end{array}
  \]

Rule “hasParent(x,y) ∧ hasParent(z,y) → siblingOf(x,z)” is expressed by “hasParent o hasParent ⊆ siblingOf” DL axiom, where hasParent is inverse property of hasParent. We transform this rule to MMP program like below:

\[
\text{meta_statement('owl': 'inverseOf'}
\begin{array}{l}
  ('f': 'parentOf', 'f': 'hasParent') \leftrightarrow \text{true}.
  \end{array}
\]
meta_statement('owl': 'propertyChainOf' (5')
('f': 'siblingOf', ['f': 'hasParent',
'f': 'parentOf']) ← true).

Rule “Man(x) ∧ siblingOf(x,y) → brotherOf(x,y)”
is transformed to MMP as follow:
meta_statement('owl': 'objectHasSelf' (6')
('f': 'Man', P_M02) ← true).

meta_statement('owl': 'propertyChainOf' (7')
('f': 'brotherOf', [P_M02, 'f': 'siblingOf']) ← true).

Rule “brotherOf(x,y) ∧ parentOf(y,z) → uncleOf(x,z)”
is transformed to MMP as follow:
meta_statement('owl': 'propertyChainOf' (8')
('f': 'uncleOf', ['f': 'brotherOf',
'f': 'parentOf']) ← true).

... We then pose some queries to the meta-interpreter and get the answers as the following:

?- demo('f': 'siblingOf'('f': 'M02', X)).
X = 'f': 'F02'.
//The adopted clauses are (ast), (acpc), (conj), (5'), (true), (ost), (1),
(mst), (acip), (4'), (2).

?- demo('f': 'brotherOf'(X, 'f': 'F02')).
X = 'f': 'M02'.
// The adopted clauses are (ast), (acpc), (conj), (7'), (true), (facc),
(6'), (mst), (2'), reasoning of siblingOf.

?- demo('f': 'uncleOf'('f': 'M02', X)).
X = 'f': 'M03'.
// The adopted clauses are (ast), (acpc), (conj), (8'), (true), reasoning
of uncleOf,(3).

5 Conclusion
In this paper we have presented a meta-logical framework for representing and reasoning with ontologies and rules expressed in OWL 2. The logical system of our framework consists of meta-programs transformed from ontologies and rules expressed in OWL 2, and an inference engine defined by a demo predicate with the extra auxiliary axioms proposed in the paper.

References: