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

Visit Hirankitti, and Trang Mai Xuan

Abstract— **OWL became a de facto language for representing Semantic Web ontologies. Having been improved and extended with rules, the language was later updated to OWL 2. In this paper we propose a framework for reasoning with an OWL2 ontology and rules using meta-logic. Our meta-logical system consists of meta-programs expressing onotologies and rules, and an inference engine in a form of meta-interpreters defined by a demo(.) predicate. The framework reasons with ontologies and rules in OWL 2 by first an ontology and rules being translated into meta-statements, and these meta-statements then being reasoned by the meta-interpreter, which provides a query answering mechanism to infer implicit information. A comparative study of related works has revealed a merit of our meta-logical representation approach that separates the meta level knowledge from the object level one.**

Index Terms— **Semantic Web Ontologies, OWL 2, Rules, Metalogic, Meta-reasoning.**

I. Introduction

Ontologies and rules have played an important role in the Semantic Web (or shortly 'SW'). An ontology forms vocabularies and sentences used to express knowledge, and this knowledge can be shared on the web. OWL was accepted by W3C as a language for representing a web ontology. Its core, OWL-DL, is essentially an XML encoding of an expressive Description Logic (DL) built upon RDF (Resource Description Framework) with a substantial fragment of RDF-Schema (RDFS). The vocabularies defined in such an ontology consist of classes (or so-called 'concepts') and properties (so-called 'roles'); in logic classes can be treated as unary predicates while properties as binary predicates, and all these predicates represent relations. OWL was successfully adopted for the semantic web in the past. However, some knowledge should be formulated more naturally as rules rather than axioms. Unfortunately, the rules are missing from OWL.

A rule representation is a formalism used in logic programming. It has been proposed as a promising form of knowledge representation in SW which complements to other means of knowledge representation in OWL. In the broadest sense, a rule can be any statement of the form "if the precondition *p* holds then the conclusion *c* holds", where the precondition and the conclusion are logical sentences. The realization of rules allows a means to deduce and combine information. This leads to a way for enhancing content, and supporting reasoning capabilities, on OWL ontologies.

The extension of SW ontologies with rules has recently attracted much attention in the Semantic Web research, and many approaches have been proposed for it. One of them is to combine DL with first-order Horn-clause rules. This is the basis of the Semantic Web Rule Language, SWRL [2], a language for rule formulation and rule extension to OWL. However, inferences on SWRL rules can lead to undecidability even though the rules are assumed to be function-free [2]. In order to make the inferences decidable, some restrictions were put upon the rule language in the form of DL-safe rules [7] or the form of Description Logic Programs (DLP) [1]. Recently, a new revision of OWL has been developed by W3C, it is called 'OWL 2'. OWL 2 has much improvement on its predecessor—OWL—especially with rule formulation based on DL $\mathcal{SROIQ}$ [11], in which DL rules can be completely ensured to be decidable fragment of SWRL.

In our previous work [9], we have developed a meta-logical approach for reasoning with semantic web ontologies expressed in OWL. In this paper we go further by extending that framework so that it can reason with SW ontologies and rules in OWL 2.

The remainder of the paper is organized as follows. Section II reviews some concepts of ontologies with a rule extension, and shows how rules can be expressed in OWL 2. Accordingly, we extend our previous framework in order to reason with ontologies and rules in OWL 2 in Section III. Section IV demonstrates how our new framework reasons with ontolologies and rules. We discuss related work in section V. Finally, section VI concludes this work.

II. Extending Ontologies with Rules

A. SW Ontologies with Rules

Adding rules to ontologies expressed in OWL could be regarded as an important step forward in the SW research, as inferences can now be performed upon SW ontologies; and many research proposals have been proposed; they range from hybrid approaches to homogeneous ones.

The idea behind the hybrid approaches was that the predicates in the rules and predicates in the ontologies are treated to be different, and suitable interface between them is provided. The research works on this direction are such as

Visit Hirankitti is with the School of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang Dist., Bangkok 10520, Thailand (e-mail: khvisit@kmitl.ac.th).
Trang Xuan Mai is with the International College, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand (e-mail: trangmx@gmail.com).

$\mathcal{AL}$-log language, a hybrid integration of Datalog and Description Logic $\mathcal{ALC}$ [4], CARIN, another hybrid integration of Datalog with different DLs [5], and a hybrid integration of OWL DL (or more precisely the DL $\mathcal{SHOIN}$) with normal rules under answer set semantics [6].

According to the homogeneous approaches both rules and ontologies are combined into the same logical language without making a priori distinction between the rule predicates and the ontology predicates. Two example languages based on 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, with Logic Programs (LP), which is the basis for rule languages. It supports a bidirectional translation of premises and inferences between the fragment of DL and LP, and vice versa. This translation enables one to construct rules on top of SW ontologies. Later SWRL was proposed in [2] as a new language for an integration of rules and ontologies, in which OWL was extended with Horn-clause rules expressed in RuleML.

Recently SWRL becomes the most widely used language for describing ontologies with rules. However, the straightforward addition of the rules to ontologies leads to undecidability when reasoning with SWRL ontologies. In order to retain decidability, some restrictions have been put upon SWRL rules, and these restricted rules can be rewritten as a set of DL axioms using features introduced in $\mathcal{SROIQ}$. This technique has been presented in [8]. Due to the fact that OWL 2 was developed based on DL $\mathcal{SROIQ}$, OWL 2 can therefore express rules in the form of DL axioms.

B. *Expressing Rules in OWL 2*

Some previous SW ontology languages such as DAML+OIL and OWL were developed based on DL $\mathcal{SHOIQ}$ [12]. $\mathcal{SHOIQ}$ provides a variety of constructors for building class expressions. The DL class expressions can be demonstrated as being corresponded to first order logic (FOL) which is used to formulate rules. Table I shows the correspondence between FOL formulae and the DL class expressions [1].

According to this correspondence, an FOL sentence can be expressed in DL as well as in DAML+OIL or OWL. For example, a rule of the form: $\mathtt{C(x)} \wedge \neg \mathtt{D(x)} \rightarrow \mathtt{E(x)} \vee \mathtt{F(x)}$ can be rewritten as a DL axiom: $\mathtt{C} \sqcap \neg\mathtt{D} \sqsubseteq \mathtt{E} \sqcup \mathtt{F}$, and a rule of the form: $\mathtt{C(x)} \wedge \mathtt{R(x,y)} \rightarrow \mathtt{E(x)}$ can be rewritten in DL as the axiom: $\mathtt{C} \sqcap \mathtt{R}.\top \sqsubseteq \mathtt{E}$. However, with some rules such as:

$\mathtt{hasParent(x,y)} \wedge \mathtt{hasBrother(y,z)} \rightarrow \mathtt{hasUncle(x,z)}$, and (A)

$\mathtt{Man(x)} \wedge \mathtt{hasChild(x,y)} \rightarrow \mathtt{fatherOf(x,y)}$. (B)

There is no correspondence so they cannot be rewritten as DL axioms, however to be able to do so we need some extra axioms in DL $\mathcal{SROIQ}$, these are some new features introduced in OWL 2.

OWL 2 which is based on DL $\mathcal{SROIQ}$ [11] supports the Role Inclusion Axiom (RIA) or so-called the "property chain" axiom and the Self concept, and these can be used to express more forms of rules, including the previous example.

Table I: DL-FOL equivalence

Expression	DL	FOL
subclassOf	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
subpropertyOf	$P_1 \sqsubseteq P_2$	$\forall x,y.P_1(x,y) \rightarrow P_2(x,y)$
transitiveProperty	$P^+ \sqsubseteq P$	$\forall x,y,z.(P(x,y) \wedge P(y,z)) \rightarrow P(x,z)$
functionalPropery	$\top \leq 1\ P$	$\forall x,y,z.(P(x,y) \wedge P(x,z)) \rightarrow y=z$
inverseProperty	$P \equiv Q^{-}$	$\forall x,y.P(x,y) \rightarrow Q(y,x)$
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf	$\neg C$	$\neg C(x)$

The Role Inclusion Axioms are the constructs of the form $\mathtt{R} \circ \mathtt{S} \sqsubseteq \mathtt{T}$ where $\circ$ is a binary composition operator. This form is equivalent to an FOL formula: $\forall x,y,z.(R(x,y) \wedge S(y,z)) \rightarrow T(x,z)$. By adopting this, rule A can easily be rewritten as a DL $\mathcal{SROIQ}$ axiom:

$\mathtt{hasParent} \circ \mathtt{hasBrother} \sqsubseteq \mathtt{hasUncle}.$

The Self concept allows one to express a "local reflexive" property, e.g. $\mathtt{R(x,x)}$, in which a role $\mathtt{R}$ relates an individual $\mathtt{x}$ to itself. The Self concept can be used to transform a property $\mathtt{R(x,x)}$ into a class $\mathtt{C}_R$ and vice versa, and this is due to a DL $\mathcal{SROIQ}$ axiom $\mathtt{C}_R \equiv \exists \mathtt{R.Self}$.

Therefore to derive the DL $\mathcal{SROIQ}$ axioms which correspond to rule B, we first transform a class $\mathtt{Man}$ into a property $\mathtt{P}_{\mathtt{Man}}$ by introducing an axiom $\mathtt{Man} \equiv \exists \mathtt{P}_{\mathtt{Man}}\mathtt{.Self}$, we then apply the previous RIA. As a result, rule B will be equivalent to the DL $\mathcal{SROIQ}$ axioms:

$\mathtt{Man} \equiv \exists \mathtt{P}_{\mathtt{Man}}\mathtt{.Self}.$
$\mathtt{P}_{\mathtt{Man}} \circ \mathtt{hasChild} \sqsubseteq \mathtt{fatherOf}.$

With the DL-FOL and DL $\mathcal{SROIQ}$-FL mappings we can express a rather wide range form of rules in DL axioms of OWL 2 and vice versa, if they satisfy certain restrictions [8]. In the next section we extend our previous framework [9] so that it can reason with ontologies and rules expressed in OWL 2.

III. Our Meta-logical Approach

A. *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 latter is a meta-interpreter, in the form of a `demo` (meta-)program, which is used to infer explicit as well as implicit information, or in other words draw conclusions, from the former. The meta-interpreter can 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.

In this paper our previous framework [9], which was designed to support reasoning with OWL ontologies, is now enhanced with an ability to reason with ontologies and rules expressed in OWL 2. Our meta-logical system can be simply illustrated in Fig. 1. In order to support rules, which are expressed by using the new features in OWL 2, the meta-program in our previous framework has to be extended with some new forms of meta-statements.

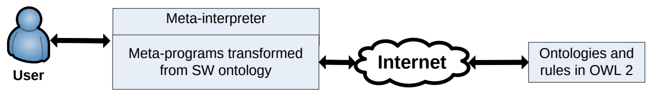
Meta-interpreter
Meta-programs transformed from SW ontology
Internet
Ontologies and rules in OWL 2
User

Fig. 1 Our meta-logical system

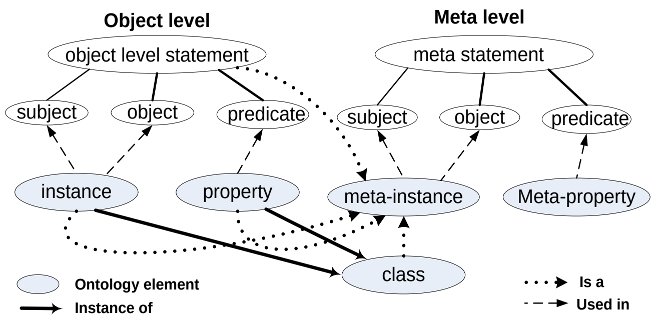

Fig. 2 Object level and Meta level of ontology elements.

To explain our framework, in the next three sub-sections 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.

B. Meta-language for an OWL2 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 <namespace>:<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. `‘←’`, `‘∧’`, `‘:’`.

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.

```
statement(
  ‘f’:‘hasSon’(‘f’:‘fa’,‘f’:‘son’)←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. `‘←’`, `‘∧’`, `‘¬’` (`‘¬’` is used to express a classical negation); 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 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 **`Pred(Sub,Obj)`**, and when it expresses a meta-level predicate stating a characteristic of a property, it has the form of **`Pred(Prop)`**, where **`Pred`** is a meta-predicate name, **`Sub`**, **`Obj`**, and **`Prop`** (a property) are meta-constants or meta-variables.

An example of a meta-term expressing a classical negation, is in the form of '¬' 'rdf':'type'(I,C), where I is a meta-constant specifying an individual, and C is a meta-constant specifying a class.

The meta-term expressing a *meta-level sentence* is a term **Pred(Sub,Obj)** or **Pred(Prop)** or a logical-connective 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) ←
   'owl':'propertyDisjointWith'(DP,P).
```

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

```
meta_statement('owl':'propertyDisjointWith'
      ('f':'likes','f':'dislikes')←true).
axiom('owl':'propertyDisjointWith'(P,DP) ←
    'owl':'propertyDisjointWith'(DP,P)).
```

C. Meta-programs of an Ontology with Rules

Each OWL2 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 the form of meta-statements for the object level: statement(P(S,O)←true). In terms of a rule system, this can be understood as a 'fact'. An example is:

```
statement('f':'hasFather'
      ('f':'M02','f':'M01')←true).
```

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

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

Some meta-statement about classes and their relationships:

```
meta_statement('rdfs':'subClassOf'
  (C,SC)←true).
```
// The class C is sub-class of class SC.

```
meta_statement('rdfs':'equivalentClass'
  (C,EC)←true).
```
// Classes C and EC are equivalent.

```
meta_statement('owl':'disjoinwith'
  (C,DC)←true).
```
// Classes C and DC are disjoint.

```
meta_statement('owl':'intersectionOf'
  (C,Cs)←true).
```
// Class C is intersection of classes in Cs.

```
meta_statement('owl':'unionOf'
  (C,Cs)←true).
```
// Class C is union of classes in Cs.

```
meta_statement('owl':'complementOf'
  (C,CC)←true).
```
// Class C is complement of class in CC.

```
meta_statement('rdf':'type'(I,C)←true).
```
// The Instance I is an instance of class C.

...

Meta-statements about properties and their relationships:

```
meta_statement('owl':'inverseOf'
  (P,IP)←true).
```
//The property P is an inversion of property IP.

```
meta_statement('owl':'symmtric'(P)←true).
```
//The property P is symmetric.

```
meta_statement('rdfs':'domain'(P,D)←true).
```
//The domain of property P is D.

...

The new features in OWL 2 that referred to in section II can be translated to the following meta-statements in **MMP**:

```
meta_statement('owl':'propertyChainOf'
  (P,[P1, P2])←true).
```
//*Express RIA: Property P is composition of properties P1, P2.*

meta_statement('owl':'objectHasSelf'(
  C, P_C)←true).

//*Express the Self concept: C is a class of individuals which are related to themselves under the role P_C.*

With such meta-statements we can transform DL rules into a meta-program. Here are examples of **MMP** that correspond to the rules we listed in section II.B:

Rule "$C \sqcap \neg D \sqsubseteq E \sqcup F$" is transformed into ***MMP****:*

```
meta_statement('rdfs':'subClassOf'
  (M,N) ←true).
meta_statement('rdfs':'unionOf'
  (N,[E,F])←true).
meta_statement('rdfs':'intersectionOf'
  (M,[C,D']])←true).
meta_statement('rdfs':'complementOf'
  (D',D)←true).
```

Rule "hasParent o hasBrother $\sqsubseteq$ hasUncle" is transformed into ***MMP****:*

```
meta_statement('owl':'propertyChainOf'
  ('f':'hasUncle',['f':'hasParent',
  'f':'hasBrother'])←true).
```

Rule "Man(x) ∧ hasChild(x,y) → fatherOf(x,y)" *is transformed into* ***MMP***:

meta_statement('owl':'objectHasSelf'
  ('f': 'Man', P_{Man}) ←true).
meta_statement('owl':'propertyChainOf'
  ('f':'fatherOf',
  [P_{Man},'f':'hasChild'])←true).

- **Meta-program for the axioms (AMP)**

AMP contains axioms for classes and properties, they are expressed in the meta-rule form. In [9], we had several axioms in **AMP** to support for OWL. In order to work with ontologies and rules expressed in OWL 2, we *add more axioms* to manipulate with the new features in OWL 2, and an axiom for a set complement. Here we list all the axioms corresponding to the formulae in Table I and to the new features in OWL 2:

```
axiom('rdf':'type'(I,C)←                      (asic)
   'owl':'subclassOf'(SC,C) ∧
   'rdf':'type'(I,SC)).
```
// Axiom to handle a subclass formula.

```
axiom(P(S,O)←                                 (acsp)
  'rdfs':'subPropertyOf'(SP,P)∧SP(S,O)).
```
// Axiom to handle a subproperty formula.

```
axiom(P(S,O) ←                                            (actp)
  'owl':'transitive'(P) ∧ P(S,O1) ∧  P(O1,O)).
```
// Axiom to handle a transitive property

```
axiom(P(S,O) ←                                            (acfp)
  'owl':'functional'(P) ∧
  P(S,O1) ∧  'owl':'sameAs'(O,O1)).
```
// Axiom to handle a functional property

```
axiom(P(S,O) ←                                            (acip)
  'owl':'inverseOf'(P,IP) ∧ IP(O,S)).
```
// Property `IP` *is an inverse property of* `P`.

```
axiom(P(S,O) ←                                            (acpc)
  'owl':'propertyChainOf'(P,[P1,P2]) ∧
   P1(S,O1) ∧ P2(O1,O)).
```
// Axiom to handle the chain property.

```
axiom(P(S,S) ←                                            (acsc)
  'owl':'objectHasSelf'(C,P) ∧
  'rdf':'type'(S,C)).
```
// Axiom to handle the Self concept.

```
axiom('rdf':'type'(I,C) ←                                 (acic)
  'owl':'intersectionOf'(C,Cs) ∧
   'intertype'(I,Cs)).
'intertype'(I,[H|T]) ←
  'rdf':'type'(I,H) ∧ 'intertype'(I,T).
```
// Axiom to handle a set intersection.

```
axiom('rdf':'type'(I,C) ←                                 (acuc)
  'owl':'unionOf'(C,Cs) ∧ 'unionType'(I,Cs)).
'unionType'(I,[H|T]) ← 'rdf':'type'(I,H).
'unionType'(I,[H|T]) ← 'unionType'(I,T).
```
// Axiom to handle a set union.

```
axiom('¬''rdf':'type'(I,C) ←                              (accc)
   'owl':'complementOf'(C, Cc) ∧
   'rdf':'type'(I,Cc)).
```
// Axiom to handle a set complement.

D. The Meta-interpreter

The meta-interpreter in our framework is constructed for reasoning 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 three 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 (including rules), and (3) `axiom(A←B)` for the mathematical axioms. The definition of `demo/1` is:

```
demo(true).                                               (true)

demo(A'∧'B) ← demo(A) ∧ demo(B).                          (conj)

demo(A) ← statement(A'←'B) ∧ demo(B).                     (ost)

demo(A) ← meta_statement(A'←'B) ∧ demo(B).                (mst)

demo(A) ← axiom(A'←'B) ∧ demo(B).                         (ast)
```

The first clause (true) is the basis for proving `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.

IV. Query Answering with Our Framework

In our framework, ontologies and rules expressed in OWL 2 are transformed into a meta-program formed by the three sub-meta-programs **MP**, **MMP** and **AMP**. This meta-program is used as the input of the meta-interpreter which is implemented in Prolog; the meta-interpreter is the inference engine reasoning with the meta-program to derive conclusions.

The family ontology [13] is used as an example to demonstrate our framework. Due to its lack of rules, three OWL2 rules are added to it. After the whole ontology is transformed into meta-programs, here are some parts of them:

- The **MP** program

```
statement('f':'hasParent'                                 (1)
  ('f':'M02','f':'M01') ← true).

statement('f':'hasParent'                                 (2)
  ('f':'F02','f':'M01') ← true).

statement('f':'hasParent'                                 (3)
  ('f':'M03','f':'F02') ← true).
…
```

- The **MMP** program

```
meta_statement('rdf':'type'                               (1')
  ('f':'M01','f':'Man') ← true).

meta_statement('rdf':'type'                               (2')
  ('f':'M02','f':'Man') ← true).

meta_statement('rdf':'type'                               (3')
  ('f':'F02','f':'WoMan') ← true).

meta_statement('owl':'complementOf'                       (4')
  ('f':'Man','f':'WoMan') ← true).

meta_statement('owl':'unionOf'                            (5')
  ('f':'Human',
   ['f':'Man','f':'WoMan']) ← true).
```

The first rule `hasParent(x,y) ∧ hasParent(z,y) → siblingOf(x,z)` added is expressed by `hasParent o hasParent`$^{-}$ ⊑ `siblingOf` DL axiom, where `hasParent`$^{-}$ is the inverse property of `hasParent`. This is transformed into the following meta-statements in **MMP**:

```
meta_statement('owl':'inverseOf'                          (6')
  ('f':'parentOf','f':'hasParent') ← true).

meta_statement('owl':'propertyChainOf'                    (7')
  ('f':'siblingOf',['f':'hasParent',
  'f':'parentOf']) ← true ).
```

The next one `Man(x) ∧ siblingOf(x,y) →  brotherOf(x,y)` is transformed into the meta-statements in **MMP**:

```
meta_statement('owl':'objectHasSelf'                      (8')
  ('f': 'Man', P_Man) ← true).

meta_statement('owl':'propertyChainOf'                    (9')
  ('f':'brotherOf',
  [P_Man,'f':'siblingOf']) ← true).
```

The third rule `brotherOf(x,y) ∧ parentOf(y,z) → uncleOf(x,z)` is transformed into the following meta-statements in **MMP**:

```
meta_statement('owl':'propertyChainOf'                    (10')
   ('f':'uncleOf',['f':'brotherOf',
   'f':'parentOf']) ← true ).
```

Now we pose some queries to the meta-interpreter to get answers as follows:

```
?- demo('rdf':'type'('f':'M02',X)).            (q1)
  X = 'f':'Man';
  X = 'f':'Human'.
```

//1st answer is supported by (mst), (1'), and (true), and 2nd answer is supported by (ast), (acuc), (conj), (5'), (1'), and (true).

```
?- demo('¬''rdf':'type'('f':'F02',X)).         (q2)
  X = 'f':'Man'.
```

//The adopted clauses are (ast), (accc), (conj), (mst), (4'), (3') and (true).

```
?- demo('f':'siblingOf'('f':'M02',X)).         (q3)
  X = 'f':'F02'.
```

//The adopted clauses are (ast), (acpc), (conj), (7'), (true), (ost), (1), (mst), (acip), (6'), and (2).

```
?- demo('f':'brotherOf'(X, 'f':'F02')).        (q4)
  X = 'f':'M02'.
```

//The adopted clauses are (ast), (acpc), (conj), (9'), (true), (acsc), (8'), (mst), (2'), and the clauses adopted for answering q3.

```
?- demo('f':'uncleOf'('f':'M02',X)).           (q5)
  X = 'f':'M03'.
```

//The adopted clauses are (ast), (acpc), (conj), (10'), (true), (3), and the clauses adopted for answering q4.

V. Related Works

We now look at other approach that enhances ontologies with rules. Grosof et al. [1] proposed the Description Logic Program (DLP); this approach supports bi-directional translation between logical sentences from DLP fragment of Description Logic and logic programs.

According to this approach, every concept referred to in an ontology is mapped into a unary relation with a concept name becoming a name of the relation and an individual name becoming an argument. Every instance-property-instance relationship is mapped into a binary relation. In addition, concepts as well as property constructor statements are converted into rules. The distinction between this approach and ours is the following.

Firstly this approach was designed to support a subset of DAML+OIL, and provides only a mapping from RDFS and DAML+OIL into logic programs, but does not support the new features in OWL 2, such as the property chain axiom and the Self concept. Secondly, this approach has a weakness when representing an ontology in a logic program. For example, to represent the statement "a is union of b_1, b_2, …, b_n", it requires n number of rules to do so, i.e. `a(X):-b1(X), …, a(X):-bn(X).` However, in our representation, this requires only one statement, that is our (acuc) axiom, which is more compact.

Even more importantly, their representation of logic program is at the object level only. Thus, the names of concepts, or the names of roles, in an ontology which are meta-terms cannot be accessed and reasoned from their logic program and therefore cannot be queried by an inference engine. For example, in their representation, statement (1) and (1') in section IV would be presented as the facts:

```
hasParent('f':'M02','f':'M01').
Man('F':'M01').
```

With these clauses we can ask only the question who is a man?, or who is a parent of `'M02'`?, but it is impossible to get answers to a question like which class `'M01'` is an instance of?, or what a relationship between `'M02'` and `'M01'` is? This is because a predicate name is a meta-level information that cannot be reasoned and queried at object-level by a Prolog interpreter. However, in our approach since we separate the meta-level from the object-level knowledge in an ontology, such queries can be asked and answered via the `demo` predicate; i.e.:

```
?-demo(P('f':'M02','f':'M01')).
  P='f':'hasParent'.

?-demo('rdf':'type'('f':'M01','f':X)).
X=Man.
```

In addition the object-level information said earlier can also be asked.

VI. 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 new extra auxiliary axioms proposed in the paper.

Acknowledgment

We gracefully acknowledge the financial support for this research from the Japan International Corporation Agency (JICA) under the AUN/SEED-Net Program.

References

[1] Benjamin N. Grosof, I. Horrocks, Raphael Volz, Stefan Decker, *Descripstion Logic Programs: Combining Logic Programs with Description Logic*, In Proc. of the 12th Int'l Conf. on the WWW, pp. 48-57, ACM, 2003.
[2] Ian Horrocks, Peter F. Patel-Schneider, *A Proposal for an OWL Rules Language*, In Proc. of the 13th Int'l Conf. on the WWW, ACM, 2004.
[3] Jakob Henriksson, Jan Małuszyński, *Hybrid integration of rules and ontologies: A constraint-based framework*, Rule and ontology integration workshop, RuleML 2006 Athens, GA, USA, 2006.
[4] Francesco M. Donini, Maurizio Lenzerini, and Andrea Schaerf, *AL-log: Integrating datalog and description logics*, Journal of Intelligent Information Systems, p. 227 – 252, 1998.
[5] Alon Y. Levy and Marie-Christine Rousset, *Combining Horn rules and description logics in CARIN*, Artificial Intelligence, pp. 165 – 209, 1998.
[6] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits, *Combining Answer Set Programming with Description Logics for the Semantic Web*, In Proc. Ninth International Conference of Principles of Knowledge Representation and Reasoning (KR2004), p. 141 – 151, 2004.
[7] Boris Motik, Ulrike Sattler, and Rudi Studer, *Query answering for OWL-DL with rules*, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 3(1)41-60, 2005.
[8] F. Gasse, U. Sattler, and V. Haarslev, *Rewriting Rules into $\mathcal{SROIQ}$ Axioms*, Poster at 21st Int. Workshop on DLs (DL-08), 2008.
[9] V. Hirankitti, and V. X. Tran, *A Meta logical Approach for Reasoning with Semantic Web Ontologies*, In proc. of the 4th IEEE Int. Conf. on Computer Sciences: Research, Innovation & Vision for the Future, Vietnam, pp. 228-235, 2006.
[10] R. A. Kowalski, J. S. Kim, *A Metalogic Programming Approach to Multi-agent Knowledge and Belief*, in AI and Mathematical Theory of Computation, pp. 231-246, 1991.
[11] I. Horrocks, O. Kutz, U. Sattler, *The even more irresistible $\mathcal{SROIQ}$*, In Proc. of the 10th Int. Conf. On Principles of Knowledge Representation and Reasoning, 2006, pp. 57-67, AAAI Press. I.
[12] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, *OWL Web Ontology Language: Semantics and Abstract Syntax*, W3C Recommendation, http://www.w3.org/TR/owl-semantics/, Feb. 2004.
[13] The family ontology, http://www.owldl.com/ontologies/family.owl.