A Meta-reasoning Approach for Reasoning with SWRL Ontologies

Visit Hirankitti, and Trang Mai Xuan

***Abstract*—SWRL is designed for enhancing inferential power on OWL ontologies by introducing rules to the language. With SWRL, rules are allowed to combine with OWL ontologies in order to support deduction on the semantic web ontologies. Earlier we have developed a meta-logical approach for reasoning with Semantic Web ontologies expressed in OWL [5] and OWL 2 [7]; with the advent of SWRL, in this paper we shall extend our framework to support OWL with rules, and hence to support SWRL. Meta-languages together with a meta-interpreter, defined by `demo(.)` predicate, are proposed and used for reasoning with Semantic Web ontologies with rules.**

***Index Terms*—Logic Programming, Meta-logic, Ontology, Rules, Semantic Web.**

I. INTRODUCTION

Ontology is an essential part of the Semantic Web (or shortly 'SW') as it forms vocabularies and statements for representing knowledge shared across the web. For an ontology to be processed automatically or reasoned logically by computers, it needs to be specified formally and declaratively. Several XML-based markup languages have been developed for expressing an ontology, they were influenced by different formalisms such as object-oriented approaches, first order logic, and Description Logic [1]. For instance, a core data representation language for SW is RDF [2]; RDF is used to represent resources in a form of subject-predicate-object triples, whereas RDF Schema (RDFS), is used to describe classes, properties and their relationships in the domain discourse, and the RDFS uses them to create a lightweight ontology; Web Ontology Language (OWL) [3] is a language derived from description logic, and offers more constructs over RDFS. OWL is used to create a more meaningful ontology, so that we can infer further information from that ontology. However, OWL still suffers from limitations due to the fact that while the language includes a relatively rich set of class constructors, its ability to describe properties is rather weak. Particularly, in OWL DL, which is based on DL $\mathcal{SHOIN}$, there is no composition constructor, so it is impossible to describe relationships between a composite property and another property. For instance, the composition of the "parent" and "brother" properties cannot be used to define "uncle" property in OWL DL. Therefore, to overcome this limitation the OWL DL needs to be extended. One way to do that is to move from OWL to OWL 2 which is an extension of OWL and is based on the more expressive Description Logic—DL $\mathcal{SROIQ}$. Another alternative is to extend OWL with a rule language, and SWRL (Semantic Web Rule Language) [4] was proposed for this purpose. SWRL provides Horn clause rule extension to OWL DL, so that an OWL ontology can now be combined with rules. Therefore, SWRL allows deduction to perform on the Semantic Web ontologies.

In our previous work [5], we have developed a meta-logical approach for reasoning with SW ontologies described in RDF, RDFS and OWL. To extend this in order to support rules in the SW ontologies, in this paper we improve our previous framework to support reasoning with SWRL ontologies.

The rest of the paper is organized as follows. In the next section we first give an overview on OWL and SWRL. Next we extend our previous framework to reason with rules in section III, and in section IV we demonstrate how our framework can reason with ontologies with rules. Later we discuss some related work in section V, and finally we conclude our work in section VI.

II. ONTOLOGY LANGUAGES

A. OWL

The Semantic Web has been designed to support automated processes and intelligent agents, so that they can access and process semantic information automatically. Semantic information, in the form of ontologies, is defined as well-formed constructs to represent concepts in a certain domain, so that intelligent agents can interpret their meaning logically. To serve for this purpose, many ontology languages such as RDF, RDFS and OWL were developed. RDF (Resource Description Framework) provides a basis data model for SW. However, RDF itself does not support a data schema. This led to the development of RDFS which provides facilities to define simple taxonomies among concepts and relations. Whilst RDFS is used to represent a simple ontology, a more expressive language for ontology representation, namely 'OWL' was later developed.

OWL has become a standard language for expressing an

Visit Hirankitti is with the School of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang Dist., Bangkok 10520, Thailand (phone: 668-0454-9990; e-mail: v_hirankitti@yahoo.com).

Trang Xuan Mai is with the International College, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand (phone: 668-7679-7235, e-mail: trangmx@gmail.com).

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

ontology. It supports several features beyond the simple definition of the hierarchies of RDFS (using rdfs:subProperty, rdfs:subClassof); in OWL we can define relations between properties and classes. More expressively, OWL allows properties to be transitive, symmetric, inverse, functional, and inversely functional. OWL also supports an ability to define complex classes in terms of logical combinations (e.g. union) of other classes. Furthermore, in OWL we can state which objects (individuals) belong to which class, and what the property values are for the specific individuals. Equivalence properties can be asserted on classes and properties, disjoint properties can be asserted on classes, as well as equality and inequality properties can be asserted between individuals.

Although OWL seems to be a very expressive language for describing an SW ontology, it needs however to embrace rules to allow deduction to perform on the ontology.

B. SWRL

Whist OWL provides rich vocabulary needed for expressing an ontology, but this ontology has limitation on inferential power related to composite properties as mentioned earlier. Therefore, adding rules to OWL will make it a more viable language for ontology representation. The basic idea for OWL rules is to extend OWL with a form of rules while maintaining maximum backward compatibility with OWL's existing syntax and semantics; and RuleML (Rule Markup Language) [8] was adopted as the language to express such a rule. Later, a new language SWRL, which is the result of combining OWL DL—the sublanguage of OWL—and RuleML, was introduced.

SWRL rules are in the form of implication consisting of an antecedent and a consequent, where description-logic expressions can occur in both. The intended interpretation of the rule corresponds to that in the classical first-order logic, that is to assert the rule, whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold; so an SWRL rule is in this form:

$$Antecedent \rightarrow Consequent.$$

Both antecedent and consequent of a rule consist of one or more *atoms*. With homogeneous combination of OWL and RuleML, the *atoms* can be in the form of either C(x), P(x,y), Q(x,z), sameAs(x,y), differentFrom(x,y), or builtIn(*pred*,z_1,…,z_n), where C is an OWL DL description, P is an OWL DL *individual-valued* property, Q is an OWL DL *data-valued* property, *pred* is built-in relation, x and y are either *individual-valued* variables or OWL individuals, and z, z_1, …, z_n are either *data-valued* variables or OWL data literals.

For example, a rule for expressing "uncle" relation can be written as follows:

$$hasParent(x,y) \wedge hasBrother(y,z) \rightarrow hasUncle(x,z).$$

A rule with multiple atoms in consequent can be transformed into multiple rules. That is, let the multiple atoms in the antecedent form a conjunction *B*, and multiple atoms in consequent form a conjunction $H_1 \wedge H_2$. We can equivalently express one rule of the form of $B \rightarrow H_1 \wedge H_2$ by the two rules $B \rightarrow H_1$ and $B \rightarrow H_2$ (due to $B \rightarrow H_1 \wedge H_2 \equiv B \rightarrow H_1 \wedge B \rightarrow H_2$).

SWRL abstract syntax was defined by adding axioms to OWL semantics and its abstract syntax [9] in order to allow the rule axioms, and the syntax of the rule axiom is:

axiom ::=rule
rule ::='Implies('{annotation} antecedent consequent)'
antecedent ::= 'Antecedent('{atom}')'
consequent ::= 'Consequent('{atom}')'.

For example, the rule "uncle" can be included to an ontology by adding the following rule axiom to that ontology.

Implies(Antecedent(hasParent(x,y) hasBrother(y,z))
Consequent(hasUncle(x,z))).

SWRL provides a rule extension to OWL, and this allows rules to be represented with OWL ontologies, and these are to be reasoned by our framework in the next section.

III. Our Framework

A. A Meta-logical approach

In our approach we applied logic programming in the context of meta-logic [11] to SW. 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 SWRL ontology at the meta-level. That is, an ontology with rules 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 can be extended to communicate to the Internet to obtain SW ontologies and rules from a web site, communicate with other agents to exchange SW information [6], and draw inference consequences from SW ontologies for the user.

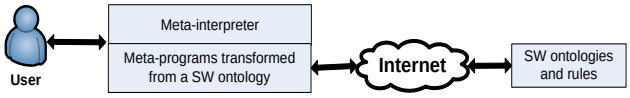

Fig. 1 Our meta-logical system

To explain our framework, in the following sub-sections we first introduce our meta-language used for formulating the meta-programs of SWRL ontologies, and then describe the meta-programs in details. Finally we describe our meta-interpreter.

B. Meta-language for an SWRL 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

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

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'`, `'disjointWith'`. 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.

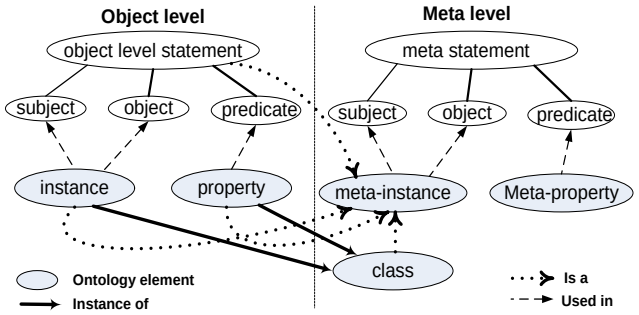

Fig. 2. Object level and Meta level of ontology elements.

According to our framework, in an SW ontology we distinguish between its object and meta information, 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. Since both object-information and meta-information are to be reasoned (and probably manipulated) by a meta-interpreter, only *syntactic forms* are processed by it. Therefore, a meta-representation of the two kinds of meta-statements is required by the meta-interpreter in order to reason with them. This is the reason why both object-information and meta-information statements have to be expressed in yet another meta-program.

Object level information and meta-level information of an SWRL ontology are expressed in our meta-languages. According to these two levels, we classify meta-language elements into two groups: one discussing mainly about objects, their relationships, and SWRL rules, 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')`. The meta-term expressing an *object-level sentence* is a term **P(S, O)** or a logical-connective function symbol applied to the tuple of these terms. We presume all meta-variable appearing in the object-level sentence are universally quantified. One form of this sentence is a Horn-clause, e.g.

```
'f':'hasFather'(Ch,F)'←''f':'hasParent'(Ch,F)'∧'
                'rdf':'type'(F,'f':'Man').
```

The meta-term, expressing an object-level predicate, is equivalent to a form of Horn-clause with an empty body. Thus, we can put `true` instead of the emptiness in its body, e.g.

```
'f':'hasSon'('f':'fa','f':'son') '←' true.
```

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has a form *statement(object-level-sentence)*, note that this statement is used to represent a Horn-clause rule translated from an SWRL rule, e.g.

```
statement('f':'hasSon'
 ('f':'fa','f':'son') '←' true).

statement('f':'hasFather'(Ch,F) '←'
   'f':'hasParent'(Ch,F)'∧'
   'rdf':'type'(F,'f':'Man')).
```

- **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. `'←'`, `'∧'`; or `':'`; or a set operator 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, disjoint with, etc.
Class-instance relations: instance of, class of, etc.
Property-property relations: subproperty of, 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, label.
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 **`Pred(Sub,Obj)`**, and when it

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

expresses a meta-level predicate stating a characteristic of a property, it has the form **`Pred(Prop)`**, where **`Pred`** is a meta-predicate name, **`Sub`**, **`Obj`**, and **`Prop`** (a property) are meta-constants or meta-variables.

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. We treat the sentence as a Horn-clause **meta-rule**, for the empty tuple in the body we put `true` there instead, e.g.

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

'owl':'propertyDisjointWith'(P,DP) ←
   'owl':'propertyDisjointWith'(DP,P).
```

Meta-statement being a meta-representation of a meta-level sentence accessible by our meta-interpreter. It has two forms *meta_statement(meta-level-sentence)* and *axiom(meta-level-sentence)*, the latter presents a rule for a mathematical axiom, and a built-in atom in SWRL, e.g.

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

axiom('owl':'propertyDisjointWith'(P,DP) ←
   'owl':'propertyDisjointWith'(DP,P)).

axiom('swrlx':'builtinAtom'('lessThan',x,y) '←'
    builtin(x < y)).
```

C. Meta-programs of an SWRL Ontology

To formulate meta-programs from SW ontologies to use in our framework, each SWRL 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 in **MML** called **AMP** is also needed for the inference engine, i.e. our meta-interpreter, to reason with **MP** and **MMP**.

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

MP contains information about instances and their relationships in terms of meta-statements for the object level: `statement(P(S,O) ← true)`, and `statement(P(S,O) ← Body)`, where `Body` is either single object-level predicate or conjunction of object-level predicates. The latter form expresses a Horn-clause rule. Here is an example of **MP**:

```
statement('f':'hasParent'
      ('f':'fa', 'f':'son') ← true).

statement('f':'hasUncle'(X,Z) ←
   'f':'hasParent'(X,Y) ∧ 'f':'hasBrother'(Y,Z)).
```

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

MMP contains meta-statements for classes, properties, their relationships, and class-instance relations in terms of meta-rules. The **MMP** is represented in the following forms:

```
meta_statement(P(S,O) ← true),
meta_statement(P(S,Os) ← true), and
meta_statement(C(Prop) ← true),
```

where `P`, `S`, `O` are predicate, subject, and object of a triple (S, P, O) defined in the ontology. `C` is a characteristic of a property `Prop`. `Os` is a tuple composing of several objects.

Here is a typical example of **MMP**:

```
//meta-statement about classes and their relationships
meta_statement('rdfs':'subClassOf'(C,SC) ← true).
meta_statement('owl':'disjoinwith'(C,DC) ← true).

//meta-statements about properties and their relationships
meta_statement('owl':'inverseOf'(P,IP) ← true).
meta_statement('owl':'symetric'(P) ← true).
…
```

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

AMP contains axioms for classes and properties, they are expressed in the meta-rule forms. In addition, **AMP** also contains axioms for built-in atoms in SWRL, and the purpose of introducing these axioms is to provide a way to translate SWRL built-in atoms in into the corresponding Prolog atoms with matched built-in predicates. Here is a typical sample of **AMP**:

```
axiom('owl':'equivalentClass'(C,EC) ←                    (asec)
      'owl':'equivalentClass'(EC,C)).

axiom('owl':'inverseOf'(P,IP) ←                          (asip)
      'owl':'inverseOf'(IP,P)).

axiom(P(S,O) ←                                           (acip)
      'owl':'inverseOf'(P,IP) ∧ IP(O,S)).

axiom(P(S,O) ← 'owl':'symmetric'(P) ∧ P(O,S)).           (acsmp)

axiom('swrlx':'builtinAtom'('lessThan',x,y) ←            (albia)
      builtin(x < y)).

axiom('swrlx':'builtinAtom'('equal',x,y) ←               (aebia)
      builtin(x = y)).
…
```

D. The Meta-interpreter

The meta-interpreter in our framework is used 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 the meta-interpreter can infer an answer `A` from the meta-programs. The interpreter is defined by adapting the Vanilla meta-interpreter [11] for reasoning with the meta-programs, which transformed from SWRL ontologies, where we have identified 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, and (3) `axiom(A ← B)` for a supporting mathematical axiom. The definition of `demo/1` is:

```
demo(true).                                              (true)
demo(A'∧'B) ← demo(A) ∧ demo(B).                         (conj)
demo(builtin(BI)) ← BI.                                  (btin)
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 basic case for proving an atom `true`. The second clause (conj) is used for proving a conjunctive goal. The third clause (btin) is used for

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

translating SWRL built-in atoms into its corresponding Prolog atoms with matched built-in predicates. The last three clauses (ost), (mst), and (ast) are used for proving three meta statements from the three meta-programs **MP**, **MMP** and **AMP** respectively.

IV. Query Answering with Our Framework

With our framework, SWRL ontologies are transformed into meta-programs **MP**, **MMP**, and **AMP**. The meta-programs are inputs to the meta-interpreter and they are all implemented in Prolog. Then the meta-interpreter is used to derive conclusions from the meta-programs.

We use the family ontology taken from [10] for a demonstration purpose of our meta-interpreter. This is an SWRL ontology, and rules are expressed by the SWRL syntax. Firstly, the ontology is transformed into meta-programs, here we show some parts of them:

- The **MP** program

```
statement('f':'hasParent'                                          (1)
     ('f':'M02','f':'M01') ← true).
statement('f':'hasParent'                                          (2)
     ('f':'M02','f':'F01') ← true).
statement('f':'hasParent'                                          (3)
     ('f': 'M03','f':'M02')  ← true).
statement('f':'hasParent'                                          (4)
     ('f': 'M05','f':'M02') ← true).
statement('f':'hasParent'                                          (5)
   ('f':'F02','f':'M05') ← true).
statement('f':'hasAge'('f':'M02',25) ← true).                      (6)
```

//Statements expressing Horn-clause rules

```
statement('f':'hasFather'(C,F) ←                                   (r1)
   'f':'hasParent'(C,F) ∧ 'rdf':'type'(F,'f':'Man')).
statement('f':'hasMother'(C,M) ←                                   (r2)
'f':'hasParent'(C,M) ∧ 'rdf':'type'(M,'f':'Woman')).
statement('f':'hasSibling'(P1,P2) ←                                (r3)
   'f':'hasParent'(P1,P3) ∧ 'f':'hasParent'(P2,P3) ∧
   'owl':'differentFrom'(P1,P2)).
statement('f':'hasBrother'(P,B) ←                                  (r4)
'f':'hasSibling'(P,B) ∧ 'rdf':'type'(B,'f':'Man')).
statement('f':'hasUncle'(P1,P2) ←                                  (r5)
'f':'hasParent'(P1,P3) ∧ 'f':'hasBrother'(P3,P2)).
statement('f':'hasSon'(P,C) ←                                      (r6)
'f':'hasChild'(P,C) ∧ 'rdf':'type'(C,'f':'Man')).
statement('rdf':'type'(P,'f':'Adult') ←                            (r7)
'f':'hasAge'(P,A) ∧
'swrlx':'builtinAtom'('lessThan',18,A)).
  …
```

- The **MMP** program

```
meta_statement('owl':'inverseOf'                                   (1')
     ('f':'hasChild','f':'hasParent') ← true ).
meta_statement('rdfs':'subPropertyOf'                              (2')
     f':'hasFather','f':'hasParent') ← true).
meta_statement('rdfs':'subPropertyOf'                              (3')
     f':'hasMother','f':'hasParent') ← true).
meta_statement('owl':'symmetric'                                   (4')
     ('f':'hasSibling') ← true).
meta_statement('rdf':'type'                                        (5')
    ('f':'M01','f':'Man') ← true).
meta_statement('rdf':'type'                                        (6')
     ('f':'M02','f':'Man') ← true).
meta_statement('rdf':'type'                                        (7')
     ('f':'M03','f':'Man') ← true).
meta_statement('rdf':'type'                                        (8')
     ('f':'M05','f':'Man') ← true).
meta_statement('rdf':'type'                                        (9')
     ('f':'F01','f':'Woman') ← true).
```

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

```
?- demo('f':'hasChild'('f':'M01',X)).
  X = 'f':'M02'.
```
//The adopted clauses are (acip), (ast), (1'), (mst), (1), (ost), and (true) .

```
?- demo('f':'hasSon'('f':'F01',X)).
  X = 'f':'M02'.
```
//The adopted clauses are (r6), (ost), (conj), (acip), (ast) , (1'),(mst), (2), (6'), and (true).

```
?-demo('f':'hasFather'('f':'M02',X)).
    X = 'f':'M01'.
```
//The adopted clauses are (r1), (ost), (conj), (1), (5'), (mst) and (true).

```
?-demo('f':'hasMother'('f':'M02',X)).
  X = 'f':'F01'.
```
//The adopted clauses are (r2), (ost), (conj), (2), (9'), (mst) and (true).

```
?-demo('f':'hasBrother'('f':'M03',X)).
  X = 'f':'M05'.
```
//The adopted clauses are (r3), (r4), (ost), (conj) (3), (4), (8'), (mst) and (true).

```
?-demo('f':'hasUncle'('f':'F02',X)).
  X = 'f':'M03'.
```
//The adopted clauses are (r3), (r4), (r5), (ost),(conj), (5), (3), (4), (7'), (mst), (true).

```
?-demo('rdf':'type'(X,'f':'Adult')).
  X = 'f':'M02'.
```
//The adopted clauses are (r7), (ost), (conj), (6), (albia), (ast), (btin), and (true).

V. Related Works

The SW research community has addressed similar issues and problems concerning SW ontologies and rules as that also happens in the area of logic programming. So the exchange of idea between these two research areas is inevitable.

For instance, Laera et al. [12] proposed SweetProlog as a system for translating an OWL ontology and rules into a Prolog program. It is achieved by the translation of an OWL ontology described in Description Logic and rules expressed in OWLRuleML into a set of facts and a set of rules in Prolog respectively. Then any reasoning on these facts and rules can be performed by the Prolog interpreter.

Comparing this with our work, according to their approach an OWL and RuleML ontology is entirely translated into Prolog facts and rules, where both object level and meta level knowledge of the ontology are mixed up and Laera et al. do not care to make the distinction between the two levels of knowledge, whilst our translation makes a careful separation between the two levels of knowledge. As a result, their SweetProlog can reason with any object level knowledge of an ontology as the way our approach does.

For example, with the same ontology that we use for the query answering in Section 4, according to their approach, the ontology would be transformed to Prolog facts as follows:

```
hasParent('f':'F02','f': 'M01').
hasBrother('f': 'M01', 'f': 'M03').
…
```

and the 'uncle' rule would be expressed by the Prolog rule:

```
hasUncle(X,Y):- hasParent(X,Z), hasBrother(Z,Y).
```

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Provided with this prolog program and a query like

```
?-hasUncle('f':'F02', X),
```

their SweetProlog would give the answer `X='f':'M03'`, which is the same answer as that given earlier by our meta-interpreter.

However, there could be queries, which ask about meta level information of this ontology, which their SweetProlog cannot give answers to, since there is some information that can be asked only at the meta level, but cannot do that at the object level.

For example, a query asking what the relation between `M02` and `M03` is:

```
?-P('f':'F02','f':'M03')
```

So, in SweetProlog, the Prolog interpreter will signal *a syntax error*, since a variable is not allowed to be used as a predicate name in a query. Here a predicate name is a meta level information that cannot be asked at the object level.

However, with a careful treatment of a separation between the object level and the meta level knowledge in our approach, such a query can be asked via the `demo` predicate as follows.

```
?-demo(P('f':'M02','f':'M03')).
```

and our meta-interpreter can give the answer:

```
P = hasUcle.
```

In the same direction as Laera et al., Samuel et al. proposed SWORIER [13], which also translates an SWRL ontology into Prolog facts and rules, and derives answers from the Prolog program using the Prolog interpreter. SWORIER suffers the same problem as SweetProlog does due to the similar approach of making no distinction between the object and meta levels. Comparing our work with theirs, in SWORIER Samuel et al. defined a set of *General Rules* in Prolog in order to formulate the OWL primitives. Here their *General Rules* serve the same purpose as our **AMP** program.

Another related work is a work on query answering for OWL-DL with rules [14]. In this work, OWL-DL was extended with DL-safe rules in order to provide deduction on an OWL-DL ontology. An undecidability problem of deduction on OWL-DL ontology with rules is solved by making some restrictions in DL-safe rules. This approach provides query answering algorithm that can handle only partial OWL-DL, since some axioms of OWL-DL, such as the transitivity axioms, are taken out from OWL-DL in order to maintain the decidability of their query answering algorithm. This work proposed a deduction method by means of a specific algorithm whereas we adopt a general purpose inference engine based on metalogic.

VI. Conclusion

We have presented a meta-logical framework for reasoning with an SWRL ontology. In this paper, our previous framework that was designed to support OWL has been extended to accommodate SWRL rules by improving the meta-languages to express Horn-clause rules and modifying the meta-interpreter so that it can work with the newly revised meta-languages.

Acknowledgment

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

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel Schneider(Eds.). *The Description Logic Hand book: Theory, Implementation, and Applications*. 2nd ed. Cambridge, 2007.

[2] W3C. The Resource Description Framework. http://www.w3.org/RDF/.

[3] P. F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation. http://www.w3.org/tr/2004/rec-owl-semantics-20040210/.

[4] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. *SWRL: A semantic web rule language combining OWL and RuleML*. URL, 2009. http://www.w3.org/Submission/SWRL/.

[5] 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, 2006.

[6] V. Hirankitti, and V. X. Tran. *A Meta-logical Approach for Multi-agent Communication of the Semantic Web Information*. In proc. of the 16th International Conference on Application of Declarative Programming and Knowledge Management, Lecture Notes in Computer Science; Vol. 4369, Springer-Verlag, pp. 215-228, 2006.

[7] V. Hirankitti, and M. X. Trang. *A Meta-logical Approach for Reasoning with an OWL 2 Ontologies*. In proc. of the 2010 IEEE Int. Conf. on Computer Sciences: Research, Innovation & Vision for the Future, 2010.

[8] Rule Markup Language, URL, http://www.ruleml.org/.

[9] Masahiro Hori, Jerome Euzenat, and P. F. Pate-Schneider. *OWL Web Ontology Language XML Presentation Syntax*. URL, 2003. http://www.w3.org/TR/owl-xmlsyntax/.

[10] The family ontology, URL, http://protege.cim3.net/file/pub/ontologies/ family.swrl.owl/family.swrl.owl/.

[11] 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.

[12] L. Laera, V. A. M. Tamma, and G. Semeraro. *SweetProlog: A System to Integrate Ontologies and Rules*. In Proc. Of RuleML'04, pp. 188-193, Springer Verlag, 2004.

[13] K. Samuel, L. Obrst, S. Stoutenberg, K. Fox, P. Franklin, A. Johnson, K. Laskey, D. Nichols, S. Lopez, and J. Peterson. *Translating OWL and Semantic Web rules in to Prolog: Moving toward description logic programs*. Journal Theory and Practice of Logic Programming, pp. 301-322, 2008.

[14] B. Motik, U. Sattler, and R. Studer *Query Answering for OWL-DL with Rules*. In Proc. ISWC2004, pp. 549-563. Springer, November 2004.