Reasoning about Semantic Web in Isabelle/HOL

Yue Tang
School of Computing
National University of Singapore
Republic of Singapore
tangy@comp.nus.edu.sg

Jing Sun
Department of Computer Science
The University of Auckland
New Zealand
j.sun@cs.auckland.ac.nz

Jin Song Dong
School of Computing
National University of Singapore
Republic of Singapore
dongjs@comp.nus.edu.sg

Brendan Mahony
Information Technology Division
Defence Science and Technology
Organization of Australia
Brendan.Mahony@dsto.defence.gov.au

Abstract

Semantic Web is regarded as the next generation of the World Wide Web. It provides not only the structure of the web but also meaningful semantics for the information presented. To make Semantic Web services understandable for distributed agents, formal definitions of the ontologies and their consistencies are essential. However, the existing tools for reasoning about Semantic Web ontologies are still primitive. We believe that mature Software Engineering tools, such as theorem provers, can contribute to the reasoning phase. In this paper, we present an approach of encoding the Semantic Web ontology (DAML+OIL) into the generic theorem prover Isabelle/HOL for automatic reasoning. Furthermore, a translation tool was developed to transform Semantic Web ontologies into their extended Isabelle theories. With additional intermediate lemmas, Isabelle can be used to perform both subsumption (class) level and instantiation (instance) level reasoning of the Semantic Web ontologies.

Keywords*: Semantic Web, DAML+OIL, Theorem proving, Isabelle/HOL.*

1. Introduction

Recent research on the World Wide Web have extended to the semantics of the web content. More meaningful information is embedded into the web content, which makes it possible for intelligent agent programs to retrieve related information based on their requirements. The Semantic Web [2] approach proposed by the World Wide Web Consortium (W3C) attracts the most attention. It is regarded as the next generation of the web. A Semantic Web service is a web application developed based on the Semantic Web technology. There have been some Semantic Web services developed recently, e.g., ITTALKS [4]. Ontology is one of the important concepts in Semantic Web services. An ontology is a document defining the semantic relationships between terms used in a Web service. A Semantic Web service usually works in a distributed environment due to the nature of the web. Hence, the consistency of its ontology is essential to make the service work correctly. Because the Semantic Web technology is still a relatively new research field, it is lacking in supporting tools for reasoning about ontology consistencies. Recently, some effects have been put to meet this needs. OilEd [1] is a graphical ontology editor. Fast Classification of Terminologies (FaCT) [9] is a Description Logic (DL) classifier. FaCT is built in OilEd as a reasoner. The FaCT system works on Description Logic, which is the basis of the Semantic Web. When it is invoked, FaCT can point out possible inconsistencies and help OilEd to show the graphical class hierarchy. However, FaCT has a main restriction. It can only support conceptual level reasoning but not instance level reasoning. Hence, it is impossible for FaCT to check instantiation relations. Renamed ABox and Concept Expression Reasoner (RACER) [8] is a TBox and ABox reasoner for description logic SHIQ [10]. Similar to FaCT, RACER is implemented in Common Lisp. It is a client-server system. The front-end is called RACER Interactive Client Environment (RICE). They are connected through the socket-based TCP/IP interface. RACER has been applied to projects in Semantic Web and software engineering. It has an advantage over FaCT that it can support both conceptual level and instance level reasoning. Open World Assumption (OWA) is adopted by RACER. It means that 'what cannot be proven to be true is not believed to be false'. RACER returns `true` as long as it cannot deduce the result is `false`. This may cause incorrectness of results that RACER returns.

At the same time, there are many existing tools that can support logical reasoning very well in the software engineering domain. It is believed that the Semantic Web could become an novel application domain for software modeling. The consequent research is to encode Semantic Web ontology into a formal modeling language, then use an existing proof tool to perform automated reasoning. Recently, there are some approaches of using formal modeling tools for ontology consistency checking. Alloy is a light-weight software modeling language based on first order logic. The approach of using Alloy to check and reason about Semantic Web [6] makes it possible to support an automated ontology verification. A Semantic Web ontology is transformed from its DAML+OIL representation into an Alloy specification. Alloy Analyzer then analyzes the model to check the consistency of the ontology. This approach can support both conceptual level and instance level reasoning. But it has its own limitations. A finite scope must be provided for AA to perform analysis. Most of the time this is not a problem as long as the scope is small. However, if the Semantic Web ontology is widely distributed through the Internet, the scope could be hard to define. Thus the confidence in the result will depend on the size of the scope adopted. Z is a formal specification language based on set theory and predicate logic. Z/EVES [12] is a theorem proving tool for the Z language. One of the approaches is checking the ontology consistency using Z with its proof tool Z/EVES [5]. Unlike the approach of Alloy, it does not have a limitation on the scope. This is because Z/EVES is a theorem prover and can support large scale proofs. Hence, it provide us with confidence in the consistency of an ontology. Unfortunately it is not yet a perfect approach. Like Alloy, Z/EVES depends too much on the underlying formal modeling language Z. Semantic Web ontology has to be converted into Z/Alloy models in order to perform consistency verifications. The process of such translation may cause inconsistency as well.

In this paper, we present an approach of directly encoding the Semantic Web meta-language (DAML+OIL) into the generic theorem prover Isabelle [11] for consistency reasoning. There are three main contributions of this paper, i.e., an formal definition of the DAML+OIL semantics in the Isabelle high order logic, a transformation program for translating Semantic Web ontology into their Isabelle theories and an automatic ontology reasoning environment provided by the Isabelle theorem prover. Firstly, an Isabelle theory library is defined for the generic DAML+OIL semantics. With this theory, concrete examples of Semantic Web ontologies are expressed as its extensions. Consequently, a transformation program for translating an ontology into its corresponding Isabelle theory becomes necessary. This program is developed based on an existing Java library for DAML+OIL. It is reusable and extensible for future enhancement. Finally, based on the Isabelle proving facilities, we can perform automatic ontology consistency checking both at conceptual and instance levels.

The remainder of the paper is organized as follows. Section 2 briefly introduces the background knowledge for Semantic Web and the generic theorem prover Isabelle. Section 3 presents Isabelle theory definitions for DAML+OIL semantics. Section 4 introduces a program that can transform a Semantic Web ontology into its Isabelle theory representation. Section 5 demonstrates a case study on a complete ontology reasoning process: starting from transformation, followed by goal identifications, ending with additional supporting lemmas and final proofs. Examples of both class-level reasoning and instance-level reasoning are presented. section 6 concludes the paper and discusses about future work.

2. Backgrounds

2.1. Semantic Web overview

The Semantic Web [2] has been considered as the next generation of the World Wide Web. It not only includes structural and visible data like traditional WWW, but also describes data with machine-understandable semantics. In a word, web contents become readable to intelligent software agents in addition to human beings.

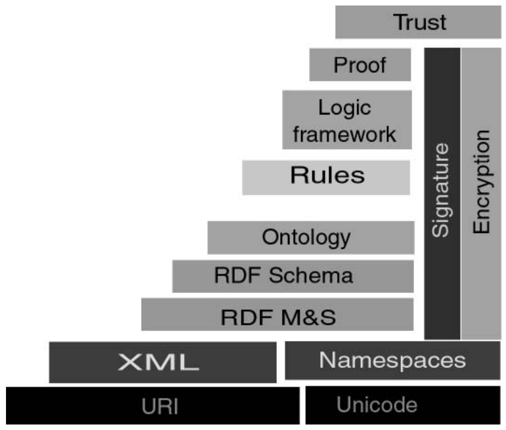

Figure 1. Semantic Web layers proposed by Tim Berners-Lee

Figure 1 shows the structure of the Semantic Web layers, a vision from Tim Berners-Lee, the creator of the WWW. It illustrates a clear picture of relations between all layers. We put our focus on the ontology (DAML+OIL) related layers, i.e., from the RDF and RDFS up to the logic framework and proof layers (in Isabelle).

2.1.1. RDF and RDFS

In order to fulfill the requirements of understandable data in the Semantic Web, ontologies are defined and distributed throughout the World Wide Web. An ontology can be viewed as a kind of XML document containing the formal definitions of terminologies and their relations. The Resource Description Framework (RDF) [7] forms a foundation for intelligent agents to exchange information. It provides a framework for meta-data processing. An RDF statement is a logic triple that has the form: `<subject property object>`, where `subject` refers to a Web resource, `property` and `object` describe one of its properties (relations) with other web resources. Hence, in contrast to a simple XML document, an RDF document is more semantically meaningful. The RDF Schema (RDFS) [7] provides a vocabulary for RDF documents in addition to their RDF model and syntax. In addition, RDFS also puts constraints on RDF structures. Therefore, it can help software agents to explicitly 'understand' (extract) the information in a Semantic Web document.

2.1.2. DAML+OIL

The DARPA Agent Markup Language (DAML) [7] is a web ontology language based on description logic. Combined with the Ontology Interchange Language (OIL) [3], DAML+OIL [13] is developed on the basis of XML and RDFS with well-defined semantics. It provides more logic constructs for defining classes and properties as well as instances, domains and ranges of properties. It is much more powerful than RDFS on modeling Semantic Web ontologies.

The following is an example ontology in DAML+OIL. It defines a web resource `Person` with relations and restrictions. `Animal` is a class and `hasParent` is a property. `Person` is a subclass of `Animal` and has the property `hasParent`.

```
<daml:Class rdf:ID="Person">
 <rdfs:subClassOf rdf:resource="#Animal"/>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#hasParent"/>
   <daml:toClass rdf:resource="#Person"/>
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
```

With the DAML+OIL language, constraints between different web resources can be specified to a deeper content. Thus logical relationships in the ontology can be made more machine understandable to their semantic web services.

2.2. Isabelle overview

Isabelle [11] is a generic proof system for implementing logical formalisms. It provides powerful mechanisms in defining hierarchical logic theories (object logics). New object logics can be built from Isabelle metalogic, by means of constructing and proving new theories. Its fundamental inference techniques are based on higher order unification and term rewriting.

2.2.1. Isabelle theory library

Isabelle supports library definitions. A library is called `theory` in Isabelle. It can be either pre-defined or user-defined. Isabelle is released with a wide range of pre-defined logic theories, such as First Order Logic (FOL), Higher Order Logic (HOL), Zermelo Frankel set theory (ZF), Constructive Type Theory (CTT), the Logic of Computable Functions (LCF), and so on. In addition, users can also define their own theories, e.g., `MyTheory.thy`. The main content of a theory file can include type declarations, function definitions, lemmas and proofs.

2.2.2. Definitional modeling using Isabelle/HOL

Isbelle/HOL [11] implements the Isabelle high order logic library. Some of the building blocks in developing Isabelle logics are as follows.

- **Theory definition** — New theories are defined as an extension on the existing theories. Thus definitions and lemmas and proof rules can be inherited. The general form of the theory definition command is:

  `theory MyTheory = theory`$_1$ `+...+ theory`$_n$

  : where `theory`$_1$`, ..., theory`$_n$ are existing supporting theories.

- **Type definition** — New types are constructed by the declaration `typedecl` or `datatype`. Isabelle supports recursive datatype definitions. The general form of a type definition is:

  ```
  typedecl typeName
  datatype myType = NAT nat
                  | TYPE2 type2
                  | TYPE3 type3
  ```

- **Function definitions** — Functions are defined by their name, inputs, outputs and their actual definitions. They provide essential information for constructing proofs. A function can be defined as follows.

IEEE COMPUTER SOCIETY

```
consts myFunction :: "nat list => nat => bool"
defs myFunction_def : "myFunction NL V ==
                        ALL n:(set NL). n=V"
```

- **Lemmas and Theorems** — The goal to be proved in a theory is defined as `lemma` or `theorem`. It is a logic statement. Theorems and lemmas can be proved by applying proof rules and tactics. The format of `lemma` and `theorem` definitions are as follows.

```
lemma myLemma1 : "P ==> Q ==> S"
      by auto
theorem myTheorem : "[|M, N|] ==> S"
        apply (rule myLemma1)
        apply (blast)
        done
```

3. Isabelle theory definition for DAML+OIL

3.1. Basic concepts

DAML+OIL is based upon RDF and RDFS with the addition of class, property and object concepts. DAML+OIL primitives are encoded into Isabelle definitions according to their semantics [7].

An Isabelle theory `SW` is defined for the semantic model of DAML+OIL meta-language using HOL as the basic logic library.

```
theory SW = Main:
```

Based on the above sematic web DAML+OIL definition, new Semantic Web ontologies can be created as extensions to the `SW` theory. For example,

```
theory OntologyExample = SW:
```

Everything in the Semantic Web context is a type of resource. It can be classified as a `class`, a `property` or an instance `resource` type. To specify these types and the relationship between them, three axiomatic types are declared and one data type `SWresource` is created as follows.

```
typedecl class
typedecl property
typedecl resource
datatype SWresource = Class class
                    | Property property
                    | Resource resource
```

A `class` is a kind of resource that is related to a set of instances in some category. The instances can be any type of resource, i.e., `SWresource`. The function `instanceOf` declares the relationship between a `class` and its set of instances which can be any type of Semantic Web resource. Given a `class`, this function returns a set of `SWresource` that represents the instances of the `class`.

```
consts instanceOf :: "class => SWresource set"
```

A `property` also has a basic relation called `subVal`. It declares the relationship between a `property` and a set of subject-value pairs. Given a `property`, this function returns a set of `SWresource` pairs.

```
consts subVal ::
    "property => (SWresource * SWresource) set"
```

3.2. Class elements

A class elements mainly describe the relations among classes in DAML+OIL. In this section, four commonly used relations are encoded into Isabelle, i.e., `subClassOf`, `disjointWith`, `sameClassAs` and `disjointUnionOf`.

3.2.1. subClassOf

The `subClassOf` function describes the relation between two classes. A class `C1` is subclass of another class `C2` if and only if all `C1`'s instances are also inside class `C2`. The declaration and definition of `subClassOf` are as follows.

```
consts subClassOf :: "class => class => bool"
                     (infixl "[<]" 65)
defs subClassOf_def: "C1 [<] C2 ==
   (instanceOf C1) \<subseteq> (instanceOf C2)"
```

where the '`infixl`' statement defines a binary symbol '`[<]`' introduced for the `subClassOf` relation.

3.2.2. disjointWith

The `disjointWith` function defines that two classes `C1` and `C2` are disjoint so that they have no common set of instances.

```
consts disjointWith :: "class => class => bool"
defs disjointWith_def : "disjointWith C1 C2 ==
  (instanceOf C1) \<inter> (instanceOf C2) = {}"
```

The `sameClassAs` and `disjointUnionOf` functions are defined similarly.

3.3. Property restrictions

The property restrictions focus on the relationships between web resources. In this subsection, some commonly used property restrictions are discussed.

3.3.1. toClass

According to DAML+OIL semantics, the function `toClass` constructs a class `C1` with a restriction on its instances. An instance of `C1` either has no values in property `P`, or if it has values in property `P`, the respective values must belong to another class `C2`.

```
consts
 toClass :: "property => class => class => bool"
defs
 toClass_def : "toClass P C1 C2 == ALL c1 c2.
       c1:(instanceOf C1) = ((c1,c2):(subVal P)
                       --> c2:(instanceOf C2))"
```

3.3.2. hasValue

The function `hasValue` restricts the property value for a class `C`. All the instances of class `C` have the same value of property `P`.

```
consts hasValue :: "property => class =>
                    SWresource => bool"
defs hasValue_def : "hasValue P C R == ALL c.
                     c:(instanceOf C) -->
                     {r. (c,r):(subVal P)}={R}"
```

Other property restrictions such as *hasClass, cardinality, maxCardinality, minCardinality, cardinalityQ, maxCardinalityQ, minCardinalityQ* are defined correspondingly.

3.4. Property elements

3.4.1. subPropertyOf

The function `subPropertOf` defines a relation between two properties. One property `P1` is said to be a sub-property of another property `P2` if and only if the set of subject-value pairs in `P1` is a subset of that in `P2`.

```
consts subPropertyOf ::
       "property => property => bool"
defs subPropertyOf_def : "P1 subPropertyOf P2
       == (subVal P1) \<subseteq> (subVal P2)"
```

3.4.2. domainOf

The function `domainOf` restricts the domain of a property `P`. All subjects from `P` are instances of a domain class `C`.

```
consts domainOf :: "property => class => bool"
defs domainOf_def : "domainOf P C ==
  ALL (sub,val):(subVal P).sub:(instanceOf C)"
```

3.4.3. transitiveProp

The function `transitiveProp` defines the transitivity of the subject-value pairs in a property `P`.

```
consts transitiveProp :: "property => bool"
defs transitiveProp_def :
   "transitiveProp P == ALL r1 r2 r3.
     (r1,r2):(subVal P) & (r2,r3):(subVal P)
       --> (r1,r3):(subVal P)"
```

Other property elements such as *rangeOf, samePropertyAs, inverseOf, uniqueProp, unambigousProp* are defined in a similar way.

In this section, an Isabelle encoding (`SW.thy`) of the DAML+OIL semantic web language in Higher Order Logic (HOL) is presented. We have defined a complete library of the DAML+OIL semantics in Isabelle. Due to the space limitation of the paper, only parts of the encoding are listed here. This generic theory (`SW.thy`) acts as a foundation library for modeling other user-defined Semantic Web ontologies.

4. Transform from DAML+OIL to Isabelle

In section 3, we defined an Isabelle theory (`SW.thy`) for DAML+OIL semantics. The consequent task is to develop a tool to transform a Semantic Web ontology into its corresponding Isabelle theory file. Our DAML2Isa translation tool was implemented based on the Jena semantic web toolkit developed by the HP Labs Semantic Web Research Group.

4.1. Transformation rules

The `DAML2Isa` tool implements transformation rules from DAML+OIL ontology to Isabelle theory. Every Semantic Web ontology is transformed as a new theory file in Isabelle. The theory `SW.thy` acts as a base library for the new theories. An ontology information includes class, property, restriction, etc. Resources, such as classes, properties and general resources, are transformed as constant definitions in Isabelle, while restrictions are transformed as axioms. A brief description of the translation rules are as follows.

- **Class transformation** — A DAML+OIL class is transformed to an Isabelle constant definition with `class` type. The RDF ID is extracted as the class name. Axioms are generated based on the constraints that this class has.
- **Property transformation** — A property is transformed as a constant definition with `property` type. The RDF ID is taken as the property name. Properties may have property elements, which are transformed to Isabelle axioms.
- **Restriction transformation** — Property restrictions are transformed to axiom definitions in Isabelle. The axiom name is composed of the restriction name and property name followed by other related information.

Detailed translation rules are implemented in the `DAML2ISa` transformation tool. DAML+OIL ontologies can be automatically translated from their XML representations into their corresponding Isabelle theories. In the next section, we will demonstrate a complete consistency reasoning process of a 'Human Being' ontology case study using Isabelle.

5. Reasoning about Semantic Web ontology

5.1. Reasoning procedure

Figure 2 shows the proof procedure for reasoning about DAML+OIL Semantic Web ontology in the Isabelle theorem prover. First, an ontology in DAML+OIL is input into the DAML2Isa transformation program. The DAML2Isa

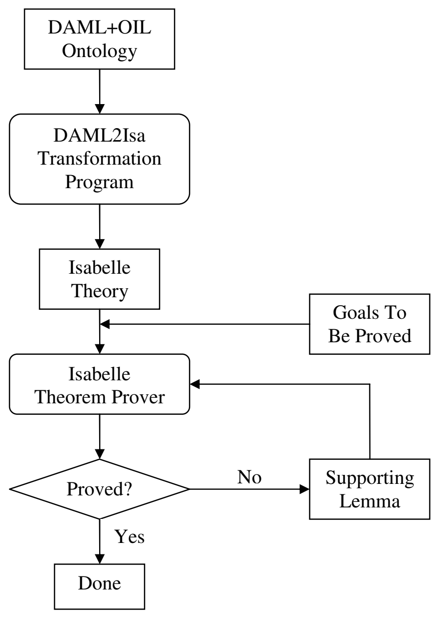

Figure 2. Ontology reasoning using Isabelle

produces an output file containing the generated Isabelle theory. After adding the goals to be proved, the file is input into Isabelle. If the goals can be proved instantly using Isabelle default tactics, it is done. Otherwise, it is necessary to add more supporting lemmas and pass the document to Isabelle again until all goals are proved.

5.2. Example ontology

A partial DAML+OIL example ontology about 'Human Beings' is used as a case study to illustrate the procedures of reasoning about Semantic Web ontology using Isabelle. This example ontology is available at the DAML web site for easy reference. The ontology defines four resource classes, `Animal`, `Person`, `Male` and `Female`, and their properties. The DAML+OIL ontology (in XML) can be translated into its corresponding Isabelle theory by the DAML2Isa tool automatically. After transforming a DAML+OIL ontology into its Isabelle theory, the next step is to identify the proof goals and apply reasoning tactics to perform the verification.

5.3. Subsumption reasoning

The purpose of subsumption reasoning is to verify the subclass relationship between two classes. This can be done through different ways of inference based on the information presented in the ontology.

5.3.1. Transitivity of a subclass

The most direct way of subsumption reasoning is based on the transitive property of a subclass relation. For example, suppose there is a class `Man` which is a subclass of `Person` and `Male`, we can infer that class `Man` is also a subclass of the `Animal` class. The goal that class `Man` is a subclass of `Animal` is defined in Isabelle as a theorem:

```
theorem
   subClassOf_Man_Animal : "Man [<] Animal"
```

In order to prove this goal, a supporting lemma is needed in the main theory `SW.thy` to express the transitivity property of a `subClassOf` function.

```
lemma
  subClass_trans : "C1 [<] C2 ==> C2 [<] C3
                            ==> C1 [<] C3"
  by auto
```

The above lemma can be easily proved using the Isabelle `auto` proof tactic. With this supporting lemma, our original goal theorem `subClassOf_Man_Animal` can be proved based on combining the other two axioms.

```
theorem
  subClassOf_Man_Animal : "Man [<] Animal"
  apply (rule subClass_trans)
  apply (rule subClassOf_Man_Male)
  apply (rule subClassOf_Male_Person)
  done
```

Note that the `apply` command denotes to apply a particular proof rule or tactic in attempts to solve a goal.

5.3.2. Class restriction on property

Another widely used method of subsumption reasoning is based on class restrictions on properties. If a class `C1` has a restriction on property `P`, and another class `C2` has the same restriction on property `P` and possibly additional restrictions, then class `C2` is a subclass of `C1`. For example, suppose a class `Adult` is subclass of `Animal` and has restriction `toClass` on property `hasParent`.

```
consts
  Adult :: class
axioms
  subClassOf_Adult_Animal : "Adult [<] Animal"
  toClass_hasParent_Adult_Person :
            "toClass hasParent Adult Person"
```

The following presents a supporting lemma for proving this goal based on functions `subClassOf` and `toClass`.

```
lemma
  toClass_subClass : "toClass P C1 C ==>
                 toClass P C2 C ==> C1 [<] C2"
  apply (unfold subClassOf_def)
  apply (unfold toClass_def)
  apply (blast)
  done
```

IEEE COMPUTER SOCIETY

Provided with the above additional lemma, the goal `subClassOf_Adult_Person` can be proved effectively as follows.

```
theorem
  subClassOf_Adult_Person : "Adult [<] Person"
  apply (rule toClass_subClass)
  apply (rule toClass_hasParent_Adult_Person)
  apply (rule toClass_hasParent_Person_Person)
  done
```

5.4. Instantiation reasoning

As stated previously, Isabelle can support instance level reasoning for Semantic Web ontologies. Instance level reasoning focuses on the relation between a Semantic Web resource and a class.

5.4.1. Membership of a class

A Semantic Web resource can be proved to be an instance of a class as long as there is enough information. For instance, based on set theory, if an element (Semantic Web resource) is an instance of a set (class), it must be an instance of any super set (super class). Here is a more complicated case. Suppose that there exist two Semantic Web resources `anAdult` and `aPerson`. The resource `anAdult` is an instance of class `Animal`, `aPerson` is an instance of class `Person`, and the resource `anAdult` has a property `hasParent` on `aPerson`. We can prove that `anAdult` is also an instance of the `Person` class.Once again, a supporting lemma is necessary for the proof. The lemma `toClassD1` explains more about the function `toClass`.

```
lemma
  toClassD1 [elim] : "toClass P C1 C2 ==>
      (c1,c2):(subVal P) ==> c2:(instanceOf C2)
          ==> c1:(instanceOf C1)"
  by (unfold toClass_def) blast
```

With the three additional axioms, the requirement for proving the theorem is fulfilled. It can be concluded that `anAdult` is an instance of the class `Person`.

```
theorem
 instanceOf_anAdult_Person :
                "anAdult:(instanceOf Person)"
 apply (rule toClassD1)
 apply (rule toClass_hasParent_Person_Person)
 apply (rule instanceOf_hasParent_anAdult_aPerson)
 apply (rule instanceOf_aPerson_Person)
 done
```

5.4.2. Non-membership of a class

Although Isabelle does not support model checking, users can still perform inconsistency checking at an instance level. For example, sometimes an inconsistency can be found by proving that a Semantic Web resource is not an instance of a certain class. As stated in the example ontology, class `Female` is disjoint with the class `Male`. Suppose there is a resource `aFemale` which is an instance of `Female`. It is obvious that `aFemale` is not an instance of `Male`. In order to prove this, a supporting lemma `disjointWithD` is necessary.

```
lemma
  disjointWithD [elim] : "disjointWith C1 C2
    ==> x:(instanceOf C1) ==> x~:(instanceOf C2)"
  by (unfold disjointWith_def) blast
```

Therefore, the final proof is as follows.

```
theorem
  notInstanceOf_aFemale_Male :
           "aFemale~:(instanceOf Male)"
  apply (rule disjointWithD)
  apply (rule disjointWith_Female_Male)
  apply (rule instanceOf_aFemale_Female)
  done
```

5.5. Instance property reasoning

Instance property reasoning is another important aspect of reasoning about Semantic Web ontologies. A Semantic Web application serves meaningful queries based on the understanding of ontology. Sometimes the necessary information is not directly stored in the database. The agent has to analyze what it knows to reply to the queries. We take a common relation between two persons as an example. Suppose a web resource `Adam hasFather Peter`. Can we conclude that `Peter hasChild Adam`? In order to answer this question, we have to define two more properties `hasFather` and `hasChild`. Let `hasFather` be a subproperty of `hasParent`, and `hasChild` be the inverse of `hasParent`.

```
consts
  hasFather :: property
axioms
  subPropertyOf_hasFather_hasParent :
               "hasFather [<<] hasParent"
consts
  hasChild :: property
axioms
  inverseOf_hasChild_hasParent :
           "inverseOf hasChild hasParent"
consts
  Adam :: SWresource
  Peter :: SWresource
axioms
  instanceOf_Adam_Person :
               "Adam:(instanceOf Person)"
  instanceOf_Peter_Person :
              "Peter:(instanceOf Person)"
  instanceOf_hasFather_Adam_Peter :
        "(Adam,Peter):(subVal hasFather)"
```

Two supporting lemmas are defined to help on the above query.

```
lemma
 rev_inverseOfD : "inverseOf P1 P2 ==>
  (sub,val):(subVal P2) ==> (val,sub):(subVal P1)"
  apply (unfold inverseOf_def)
  apply (blast)
  done
 lemma
  subPropertyD [elim] : "P1 [<<] P2 ==>
   (sub,val):(subVal P1) ==> (sub,val):(subVal P2)"
  apply (unfold subPropertyOf_def)
  apply (blast)
  done
```

By applying these two lemmas and three available axioms, the goal to prove `Peter hasChild Adam` can be reached step by step. Since `hasFather` is sub-property of `hasParent`, the fact `Adam hasFather Peter` ensures `Adam hasParent Peter`. Furthermore, `hasChild` is inverse of `hasParent`. Now we can conclude that `Peter hasChild Adam`.

```
 theorem
   instanceOf_hasChild_Peter_Adam :
                "(Peter,Adam):(subVal hasChild)"
   apply (rule rev_inverseOfD)
   apply (rule inverseOf_hasChild_hasParent)
   apply (rule subPropertyD)
   apply (rule subPropertyOf_hasFather_hasParent)
   apply (rule instanceOf_hasFather_Adam_Peter)
   done
```

In this section, we discussed the different reasoning tasks performed by Isabelle through a 'human being' ontology example. we presented each reasoning proof together with the supporting lemmas used. In our implementation, we defined a sufficient set of supporting lemmas in the `SW.thy` theory library for assisting effective reasoning.

6. Conclusion

In this paper, we demonstrate an approach of reasoning about Semantic Web ontologies using generic theorem prover Isabelle. First, an Isabelle representation of the DAML+OIL semantic web meta-language was defined, which can be used as a theory library for detailed ontology modeling in Isabelle/HOL. Furthermore, a set of supporting lemmas were developed for the purpose of effective reasoning. Second, a DAML2Isa transform tool was developed for the automatic translation from DAML+OIL ontologies into their corresponding Isabelle theory files. Third, a case study illustrated the detailed procedures for reasoning about Semantic Web ontologies in Isabelle/HOL. Three types of reasoning tasks are discussed, i.e., subsumption, instantiation and instance property reasoning. In conclusion, Isabelle can check the consistency of Semantic Web ontology at both conceptual level and instance level. At the same time, it can help answer queries that are not included in the knowledge base. In summary, we believe that the generic theorem prover Isabelle can play a contribution role in the reasoning about Semantic Web ontologies.

One part of the future work is to enhance the general Isabelle theory library `SW.thy`. It will be more effective if the theory includes not only essential functions but also a sufficient set of supporting lemmas. Isabelle is a generic theorem prover. As a result, it lacks complete automation. Hence, another part of the future work is to reduce user interactions as much as possible so that the reasoning procedure can become more efficient. As discussed earlier, model checkers and theorem provers complement to each other. To improve the performance of reasoning about Semantic Web ontologies, it could be a better solution that Isabelle is worked together with a model checker such as Alloy.

References

[1] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, pages 396–408. Springer-Verlag, 2001.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, May 2001.

[3] J. Broekstra, M. Klein, S. Decker, D. Fensel, and I. Horrocks. Adding formal semantics to the web: building on top of RDF Schema. In *ECDL Workshop on the Semantic Web: Models, Architectures and Management*, 2000.

[4] R. S. Cost, T. Finin, A. Joshi, Y. Peng, C. Nicholas, I. Soboroff, H. Chen, L. Kagal, F. Perich, Y. Zou, and S. Tolia. Ittalks: A case study in the semantic web and daml.

[5] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and beyone in Z/EVES. In *The 26th International Conference on Software Engineering (ICSE'04)*, Scotland, UK, May 2004. IEEE Press.

[6] J. S. Dong, J. Sun, and H. Wang. Checking and Reasoning about Semantic Web through Alloy. In *Proceedings of 12th Internation Symposium on Formal Methods Europe: FM'03*, Pisa, Italy, Sept. 2003. LNCS, Springer-Verlag.

[7] R. Fikes and D. L. McGuinness. An axiomatic semantics for RDF, RDF Schema, and DAML+OIL. Technical Report KSL-01-01, Knowledge Systems Laboratory, 2001.

[8] V. Haarslev and R. Möller. RACER system description. In *Proceedings of Automated Reasoning: First International Joint Conference*, pages 701–706. Siena, June 2001.

[9] I. Horrocks. The FaCT system. *Tableaux'98, Lecture Notes in Computer Science*, 1397:307–312, 1998.

[10] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[11] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[12] M. Saaltink. The Z/EVES system. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: Z Formal Specification Notation*, volume 1212 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, 1997.

[13] F. van Harmelen, P. F. Patel-Schneider, and I. H. (editors). Reference description of the DAML+OIL ontology markup language. Contributors: T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, L. A. Stein, ..., March, 2001.

IEEE COMPUTER SOCIETY