A Non-Standard Approach for the OWL Ontologies Checking and Reasoning

Yingjie Song
Dalian Maritime University, Dalian 116026, P.R. China
Email:tiantianyingjie@gmail.com

Rong Chen
Dalian Maritime University, Dalian 116026, P.R. China
Email: tsmc.dmu@gmail.com

Yaqing Liu[1,2]
[1]Dalian Maritime University, Dalian 116026, P. R. China
[2]Artificial Intelligence Key Laboratory of Sichuan Province, Zigong, China
Email: liuyaqing234@yeah.net

***Abstract*—The Semantic Web is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites. The understanding of Semantic Web documents is built upon ontologies that define concepts and relationships of data. Hence, the correctness of ontologies is vital. In this paper, we propose a new algorithm combined with the software engineering techniques, such as Alloy modeling language and its reasoner Alloy Analyzer to provide checking and reasoning service for OWL ontologies. First of all, we use Jena to parse OWL ontology documents. Next, the intermediate results are used as the inputs of the algorithms to generate the Alloy model. Futher, with the assistance of Alloy Analyzer, the Alloy model is checked. Experimental results show that this method can be carried out large-scale ontology reasoning and complex-property reasoning which are different from traditional ontology reasoning. Furthermore, the results provide useful information to guide the ontology modification.**

***Index Terms*—Ontology Reasoning, OWL, Alloy, Semantic Web**

I. Introduction

Tim Berners-Lee's [3] original vision for the Semantic Web was that information would be just as readable (and understandable) to a person or to a machine. The aim of the Semantic Web is to make Web resources more readily accessible to automated processes. Digital objects, whether web page, image, video, or some other file, would have embedded within them meta data that would provide context to the content and allow software to extract meaning from the file. To make sure that different agents have a common understanding of these, ontology is needed to describe. Basically, an ontology [11] is a collection of definitions of concepts and the shared understanding comes from the fact that all the agents interpret the concepts in the same way. The importance of ontologies in Semantic Web has prompted the development of several ontology languages. Many ontology languages have been developed, especially for the semantic web, such as OIL [12], DAML [2], DAML+OIL [6] and OWL [14]. OIL (Ontology Interchange Language) is based on three elements, namely, frame-based systems, description logics, and web standards [10]. DAML (DARPA Agent Markup Language) is developed in DARPA DAML programme. DAML+OIL is a language for expressing far more sophisticated classifications and properties of resources than RDFS. OWL, which is aimed to be the standardized and broadly accepted ontology language of the Semantic Web, is compatible with early ontology languages and provides the engineer more power to express semantics.

Reasoning with ontology languages is important to ensure the quality of an ontology. Indeed reasoning can be employed in different phases during the design, maintenance and deployment of ontology [9]. Some of the popular reasoners that are available such as RACER [22], FaCT [12], FaCT++ [21], Pellet [19]. They are all based on the description logic. For OWL, a standard ontology language for the semantic web, these reasoners and algorithms can be used to solve the reasoning problems, in particular satisfiability. When attempting to use description logic to provide reasoning services for OWL, there are many differences between OWL and description logics [1]. OWL' RDF syntax allows circular syntactic structures, in addition, the description logic does not include contain-features of OWL.

Software engineering methods and tools will be introduced in our solution. In our previous work [20], this idea has been used for DAML + OIL ontology testing, and the experiment proved that this method could work on a larger scope of property checking. This paper, the lightweight modelling language for software design Alloy [15] and its fully automatic analysis tool Alloy Analyzer [16] are used for ontology reasoning. Alloy can solve the problem which dose not exist in the description logic. It's

doi:10.4304/jcp.7.10.2454-2461

the characteristics of circular structure and contain-features that are more conducive to the expression of OWL ontologies. Alloy Analyzer is a software tool which can be used to analyze specifications written in the Alloy specification language. The Alloy Analyzer supports the analysis of partial models. As a result, it can perform incremental analysis of models as they are constructed, and provide immediate feedback to users. Alloy Analyzer is based on the new SAT-based model finder Kodkod. Kodkod applies new techniques and optimizations to the translation from relational to boolean logic (http://alloy.mit.edu/alloy4/). In our approach, the Alloy Analyzer is used to analyze the Alloy model based on the OWL ontology, and provide automatic reasoning and consistency checking for the semantic Web.

After a brief introduction and quick survey of OWL, Section 3 and 4 a simple ontology example is given which is described in OWL, after the analysis by jena, we give the algorithms which is used to transfer the results into Alloy model. Section 5 surveys some of the major problems that had to be resolved in the reasoning of ontologies with Alloy Analyzer, while Section 6 gives a simple example to illustrate the process of reasoning and the problems can be found easily. And Section 7 gives summary of the article.

II. The OWL Web Ontology Language

A. Logical Characteristic of OWL

OWL is an ontology language that has recently been developed by the W3C Web Ontology Working Group. It is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and application. OWL is defined as an extension to RDF in the form of a vocabulary entailment, i.e., the syntax of OWL is the syntax of RDF and the semantics of OWL are an extension of the semantics of RDF. In order to meet the different expressing power and computational efficiency, it comes in three flavors: OWL DL, OWL Lite and OWL FULL. OWL FULL, which contains all the language element of OWL, is the complete works of the language, and it is a syntactic and semantic extension of RDFS. OWL DL, a subset of OWL FULL, is a version of OWL with decidable inference that can be written in a frame or Description Logic manner. OWL Lite is a subset of OWL DL, and also has a lower formal complexity than OWL DL. It is the least expressive species of OWL, but it ensures the efficient reasoning.

B. Reasoners for Semantic Web

The current main reasoning tools for Semantic Web are based on description logic, such as the most widely used three well-known DL reasoners: RACER, FaCT++ and Pellet.

RACER not only can be used as a description logical system, but also can be regarded as a representation system that implements a highly optimized tableau calculus. It implements the description logic SHIQ., and facilities for algebraic reasoning including concrete domains. But RACER can only support ABox reasoning completely.

FaCT++ is the new generation of the well-known FaCT (Fast Classification of Terminologies). A new tableaux decision procedure for SHOIQ(D) is implemented. The FaCT++ system provides satisfiability testing for modal logic. In order to create a more efficient software tool and to maximize portability, it is implementated as a free open-source C++-based reasoned. However, FaCT++ can't take OWL documents directly nor any remote file. The OWL ontologies have to be translated into DIG format for the system.

Pellet is an open source, Java reasoner for OWL DL ontologies. It is based on the tableaux algorithms developed for expressive description logics. It is claimed to provide functionalities to see the species validation, check consistency of ontologies, classify the taxonomy, check entailments and answer a subset of RDOL queries.

Though these tools can reason ontologies with a high degree of automation, the complex-properties ontologies still can't be checked by them effectively. Furthermore, these reasoners can find that the ontology is error, but not be able to find out the reason.

C. Alloy and Alloy Analyzer

Alloy is a lightweight modelling language for software design, which is widely accepted as micromodels of software in the software engineering community. It is a first order relation logic and treats relations as the first element. An Alloy model consists of Signatures, Relations, Facts, Functions and Predicates. Signatures represent the entities of a system and Relations are used to describe relations between such entities. Facts and Predicates introduce constraints over such Signatures and Relations. Whereas Facts are constraints to be always valid, Predicates are named parameterized contraints for depicting operations, Functions are named expression with parameters that return results.

Alloy Analyzer is a tool for analyzing models written in Alloy. The Alloy Analyzer 4 is based on the new SAT-based model finder to support fully automated analysis of Alloy models through simulation and Assertion checking. Given a user specified scope on the model elements bounding the domain, the analyzer first translates an Alloy model into boolean formulas, and then invokes a SAT-solver to find an instance. If an instance that violates the assertion is found within the scope, the assertion is not valid and the instance is returned as a counterexample.

III. Reasoning For Owl Ontologies

This section, we will introduce the reasoning process. In the reasoning, Jena [5], Alloy and Alloy Analyzer will be used. Jena is fundamentally an RDF platform, and it supports ontology formalisms built on top of RDF, such as DAML+OIL and OWL. As is shown in Figure 1, according to Jena, we can get Class C, Property P and Statements S in the results. Next, owl2Alloy algorithm is used to translate the results into Alloy model. And then Alloy model is analyzed by the Alloy analyzer. If there

are some errors in the ontology, they must be analyzed and corrected. The modified ontology must be reasoned again.

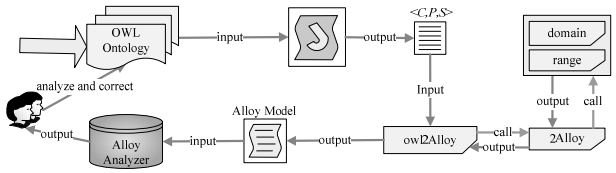


Figure1.The specific process of ontology reasoning.

A. Parsing OWL Documents

For the requirements of reasoning, classes, properties and statements are wanted. First, Jena, which is a Java framework for building Semantic Web application, is used to analyze the OWL ontology documents. We created an ontology model of *OntModel* for handing OWL ontologies, and loaded an ontology document into the model using the *read* method. The *listClasses()* method can pick out the classes of the ontology, the *listOntProperties()* method can answer an iterator over all of the ontology properties, and the *listStatements()* method can return an iterator over all the statements in this model.

There is an ontology document in Figure 2. It contains something about animal. There are four classes defined: *Animal*, *Male*, *Man* and *Female*. *Animal* is the base class, *Male* and *Female* are disjoint subclass of the base class. *Man* is subclass of *Male*. There are also three properties in the document, they are *hasFather*, *hasParent* and *hasChild*. The properties *hasParent* and *hasChild* are inverse of each other, and the property *hasFather* is subproperty of *hasParent*. We will use Jena to parse the ontology document into three sections, which are classes, properties and statements.

```
……
<owl:Class rdf:ID="Animal"/>
<owl:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</owl:Class>

<owl:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Male"/>
</owl:Class>
<owl:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Male"/>
</owl:Class>
……
<owl:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasChild">
  <rdf:inverseOf rdf:resource="hasParent"/>
</owl:ObjectProperty>
```

Figure2.The specific process of ontology reasoning.

The output of Jena contains a sequence of *Classes*, *Properties* and *Statements*, each having a couterpart in the original document. It can reflect the structure of animal ontology accurately. The three sections can mirror OWL vocabulary. Except *Classes*, *Properties* corresponding to the Class and Property of OWL, the rest of OWL language features can be converted into *Statements*. Jena read and parsed the animal ontology document, the result is shown in Figure 3.

```
<OWLClass Animal>
<OWLClass Male>
<OWLClass Man>
<OWLClass Female>
The number of Classes is：4
<OWLProperty hasFather>
<OWLProperty hasParent>
<OWLProperty hasChild>
The number of Properties is：3
[<Male>, subClassOf, <Animal>]
[<Female>, subClassOf, <Animal>]
[<Female>,disjointWith, <Male>]
[<Man>,subClassOf, <Male>]
[<hasFather>, range, <Male>]
[<hasFather>,subPropertyOf,< hasParent>]
[<hasParent>,domain, <Animal>]
[<hasParent>,range, <Animal>]
[<hasChild>,inverseOf,<hasParent>]
The number of Statements is：9.
```

Figure 3. Jena outputs of “Animal” ontology

B. The Algorithms to Generate Alloy Model

The different parts of the parsing result can be converted into different Alloy element. In Alloy, *signature* represents a set of atoms. If there are two kinds of signatures which have a father-son relation, the signature that extends another signature is said to be a subsignature of the signature it extends, and its type is taken to be a subtype of the type of the signature extended, and is declared using the in keyword *extends*. This is similar to the relation between classes, so *Classes* are converted into signatures. The Alloy universe consists of *atoms* and *relations*, although everything that you can get your hands on and describe in the language is a relation. The rest of the outputs of Jena appears in the Alloy model in the form of *relation*. As shown below, our Algorithm owl2Alloy(C, P, S, Σ) converts the input *Classes* (denoted as C), *Properties* (denoted as P) and *Statements* (denoted as S) into a textual Alloy model Σ.

Algorithm 1: owl2Alloy(C, P, S, Σ) 建议用图来表示
Input: a set C of Classes, a set P of Properties and a set S of Statements
Output：a textual Alloy model Σ
1. ← ω;
2. FOR each c∈C
3. ← Σ + 2Alloy(c, S);
4. FOR each s∈S
5. IF s.predicate.name = "disjointWith"
6. IF s.subject.subClassOf ≠ s.object.subClassOf
7. ← Σ + "pred {no c1:" + s.subjec.name +", c2:" + s.object.name +" | c1 = c2}";
8. IF s.predicate.name ="complementOf"
9. ← Σ + "pred {" + s.subject.name +"=" + C – s.object.name +"}";
10. IF s.predicate meets one of Conditions in Table 1
the corresponding generated Alloy predicate is appended to Σ
11. RETURN Σ

The owl2Alloy algorithm's input is the parsing result of Jena, and then it will generate the Alloy model Σ, which contains Alloy signatures and the relations between them. The algorithm must call the other two algorithms, 2Alloy(*c*, *S*) and domain (*p*, *S*) to complete the translation process. The top-level algorithm initializes the Alloy model Σ as an empty string ω above all. For each c, there will be a corresponding signature of the same name added to Σ, while the 2Alloy(*c*, *S*) algorithm is invoking. Next, we deal with the *Statements* throught lines 4~11. *Statements* are composed of three segments like <*subject*, *predicate*, *object*>. In the algorithm, they are represented by *s.subject*, *s.predicate* and *s.object* respectively. We mainly based the second one to determine the coming form. We use *e* to represent one of the three segments, and *e.name* to represent the name of it. The different name of predicate will generate different predicate in Σ. Lines 5~11 identify the name of each *e.predicate* to determine the predicates in the coming model Σ. If *e.predicate.name* is 'disjointWith' (lines 5~7), there will be a predicate to declare that *e.subject* and *e.object* are disjoint from each other, and there is no individual belong to both of them. Lines 8~9 are used for the condition which *e.subject* is a subclass of the complement of *e.object*. The others meet one of the condition in Table 1

The owl2Alloy algorithm also calls the other sub-algorithms, in which, the 2Alloy(*c*, *S*) algorithm is to generate a signature for a certain class *c* from the outputs of Jean. In addition to the class *c*, a set *S* of *Statements* is also used as inputs of the algorithm. At first, the first line produces a signature which is named after the input class *c*. Lines 2~4, to determine whether the class *c* is subclass of another. If *c.subClassOf* is not empty, that is to say *c* is subclass of another. We use keyword "extends" in Alloy to connect the parent-signature. From 5 to 8 lines, it comes to determine the relationship between the classes. There is a property *p*, sub-algorithm domain (*p*, *S*) is used to determine whether *c* is its domain. If it comes to this, we use range (*p*, *S*) algorithm to get the range of *p*. Then it represents with the performance of Alloy language likes "domain {property: range}". In line 9, we get the signature σ.

TABLE I.
MORE CASES FOR CONVERTING STATEMENTS TO ALLOY

	Condition	Alloy predicates
1	*s.predicate.name*="subPropertyOf"	"pred subPropertyOf{all r:"+ *s.subject.range* +"| r in "+ *s.object.range* +"}"
2	s.predicate.name=" equivalentClass "	"pred equivalentClass{ "+*s.subject*+"=" +*s.object*+"}"
3	*s.predicate.nam e=" equivalentProperty "*	"pred equivalentProperty {"+s.subject+ " = " + s.object +"}"
4	*s.predicate.name*="inverseOf"	"pred inverseOf {"+*s.subject*+"=~"+*s.object*+"}"
5	*s.predicate.name="TansitiveProperty"*	*"pred TansitivePropertyOf{ a,b,c ∈" +s.subject+" /a.("+s.predicate+") = b && b.("+s.predicate+") =c ⇒ a. ("+ s.predicate+") = c}"*
6	*s.predicate.name="hasValue"*	*"pred hasValue {#("+s.predicate.range+")=1 }"*
7	*s.predicate.name="cardinality"*	*"pred cardinality {#("+s.predicate.range+")=" + s.object"}"*
8	*s.predicate.name="maxCardinality"*	*"pred maxCardinality {#("+s.predicate.range+")<= "+s.object "}"*
9	*s.predicate.name="minCardinality"*	*"pred minCardinality {#("+ s.predicate.range+")>="+s.object "}"*
10	*s.predicate.name="SymmetricProperty"*	*"pred SymmetricProperty { a ∈" +s.predicate.domain+" && b ∈ " + s.predicate.range + " / a.("+s.predicate+") = b ⇒ b. ("+ s.predicate+") = a }"*
11	*s.predicate.name=" FunctionalProperty "*	*"pred FunctionalProperty {#("+s.predicate.range+")=1 }"*
12	*s.predicate.name=" InverseFunctionalProperty "*	*"pred InverseFunctionalProperty {#("+ s.predicate.domain + ")=1}"*
13	*s.predicate.name=" allValuesFrom "*	*"pred allValuesFrom {"+s.subject.range + " in"+ s.object " }"*
14	*s.predicate.name=" someValuesFrom "*	*"pred someValuesFrom {some r:"+s.subject.range + "| r in"+ s.object " }"*

Algorithm 2: 2Alloy(c, S, P)
Input: a Class c, a set S of Statements, a set P of Properties
Output: a signature σ
1. ← "sig " + c.name;
2. IF c.subClassOf ≠ω
3. ← σ + "extends" + c.subClassOf.name;
4. ← σ + "{";
5. FOR each p∈P
6. IF domain(p, S) = c.name
7. ← σ + p.name + " : " + range(p, S);
8. ← σ + " }";
9. RETURN σ;

The sub-algorithm domain (*p*, *S*) is used to obtain the domain of the property *p*. For a property *p*, if there is a statement *s* meet the requirement of lines 1~2, the *s.object* is considered to be the domain of *p*. Otherwise,

the domain of *p*'s parent is also its domain. We use parent (property) to represent the parent property of the parameter property, so the domain of a property is calculated recursively.

There is another sub-algorithm range (*p, S*), which is similar to the domain (*p, S)* algorithm. The algorithm aims to get the range of the property *p*.

```
Algorithm 3: domain (p, S)
Input: A property p of P, Statement S
Output: the domain of p
1. IF ∃s∈S, s.subject = p and s.predicate = "domain"
2. p.domain = s.object.name;
3. ELSE
4. p.domain = domain(parent(p), S);
5. RETURN p.domain;
```

As previously stated, the Animal ontology is taken as an example to illustrate how these algorithms are used.

We use the outputs of Jena as the inputs. Then we will get an Alloy model as is shown in Fig.4. In the model, there are four signatures which are *Animal*, *Female*, *Male* and *Man* corresponding to four classes in the inputs. The paternity relations between them can be described as follow: Female and Male both extend from Animal signature, while Man extends from Male signature. There are three properties which are *hasParent*, *hasFather* and *hasChild*. They are also the ontology's properties. The inverseOf and subPropertyOf are two predicates in the model which are converted from the statements. The former is used to illustrate the *hasParent* is the inverse of the *hasChild*. The latter show that the range of *hasFather* is subset of the range of *hasParent*.

```
sig Animal{
   hasParent:Animal,
   hasFather:Male,
   hasChild:Animal
}
sig Female extends Animal{}
sig Male extends Animal{}
sig Man extends Male{}
pred inverseOf{hasParent=~hasChild}
pred subPropertyOf{all a:Animal|a.hasFather in a.hasParent}
```

Figure4. The generated Alloy model of "Animal" ontology.

C. Verifying Ontologies with the Alloy Analyzer

Ontology reasoning is an important way to ensure the correctness of ontologies. The existing tools are mainly for the conceptual-level reasoning. Alloy, in addition to meeting the conceptual-level reasoning, can also be used as a reasoned for the instance-level. In order to prove the satisfiability of the ontology model, at least one instance of the model is needed. When Alloy Analyzer is used to analyze the model, if the model is right, it will give some instances which meet the model at random. Else, a counterexample will be given to prove that the model is inadequate.

There are several tasks to do in the reasoning, which are consistency checking, subsumption reasoning and implication relation checking. If an ontology is inconsistent, then any erroneous conclusion may be deduced by software agents. Subsumption reasoning can be described as follow: if a class C_1 is more general than another one C_2, it subsumes that C_1 can be contained by C_2. The implication relation can be used to get some implicated conclusion. All these tasks（是指什么任务 What does it mean "all these tasks can be converted"?）can be converted into Alloy assertion. In the reasoning, Alloy Analyzer will determine whether these assertions are met. If all of these have been met, then Alloy Analyzer will give some instances. Otherwise, the assertions which have not been satisfied will be pointed out. And Alloy Analyzer will generate a counterexample. All these can be used to search the root of the problem. In the following, we will give an example to illustrate the process of ontology reasoning.

In Fig.5, there is a segment of an Animal ontology. In the document, there are three classes: *Animal*, *Male*, *Female* and *Woman*. *Male* and *Female* both extend from *Animal* and disjoint with each other. *Woman* is a subclass of *Female*. The *Animal* class has a property named *animalHasFather* whose range is *Male*.The *femaleHasFather* is another property of the ontology. Its domain is *Female*, while its range is *Male*. The property *femaleHasFather* is the subPropertyOf *animalHasFather*.

```
……
<owl:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="Animal"/>
</owl:Class>
<owl:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="Animal"/>
  <owl:disjointWith rdf:resource="Male"/>
</owl:Class>
< owl:Class rdf:ID="Woman">
  <rdfs:subClassOf rdf:resource="Female"/>
</owl:Class>
……
<owl:ObjectProperty rdf:ID="animalHasFather">
  <rdfs:domain rdf:resource="Animal"/>
  <rdfs:range rdf:resource="Male"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="femaleHasFather">
<rdfs:subPropertyOf rdf:resource="animalHasFather"/>
  <rdfs:domain rdf:resource="Female"/>
  <rdfs:range rdf:resource="Woman"/>
</owl:ObjectProperty>
……
```

Figure 5. An example of error ontology.

The Animal ontology document can be converted into a Alloy module using our method. In the module, there are four signatures, two relations and a predicate. Then Alloy Analyzer is used to check the module.

As is shown in Fig.6, the editor panel of the user interface contains the Alloy models, and if there are some errors, they will be highlighted during model compilation. In the Animal model, the keyword 'in' is highlighted. The message panel displays the results of analysis. It shows that the left type and right type of highlighted word are always disjoint. Because the range of *animalHasFather* is *Male* and the range of *femaleHasFather* is *Woman*. And there is a *subProperty* constraint between *femaleHasFather* and *animalHasFather*, which mean that the *Woman* is subClassOf *Male*. It is a conflict.

Figure 6. Checking result of Alloy Analyzer.

According to the error message, we can find that the range of *femaleHasFather* property is not appropriate. It is modified into *Male* and the model is checked again. The Alloy Analyzer gives an instance for the modified model as shown below.

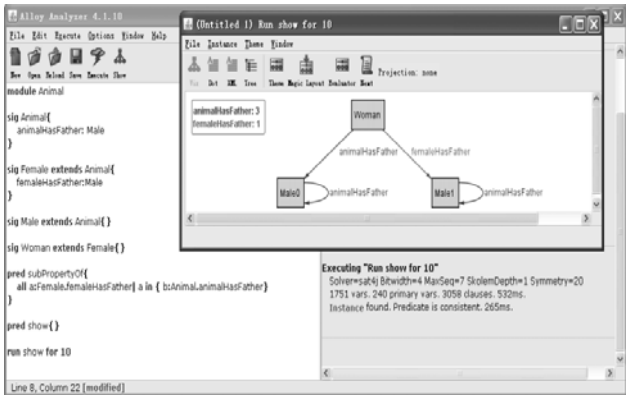


Figure 7. The modified model.

IV. RELATED WORK

Ontologies are set to play a key role in the Semantic Web by providing a source of shared and precisely defined terms that can be used in descriptions of web resources. Reasoning over such descriptions will be essential if web resources are be more accessible to automated processes.

A number of ontology inference engines, such as FaCT, RACER, and FaCT++ have been developed with the advancement of ontology languages to facilitate ontology creation, management, verification, merging, etc.. They can make explicit information from knowledge and data. But complex ontology-related properties cannot be supported by them. FaCT (Fast Classification of Terminologies) is a DL classifier that can also be used for modal logic satisfiability testing. The FaCT system includes two reasoners, one for the logic SHF (ALC augmented with transitive roles, functional roles and a role hierarchy) and the other for the logic SHIQ (SHF augmented with inverse roles and qualified numver restrictions), both of which use sound and complete tableaux algorithms. The RACER system is a knowledge representation system that implements a highly optimized tableau calculus for a very expressive description logic. It offers reasoning services for multiple TBoxes and for multiple ABoxes as well. The system implements the description lodib ALLQHIR$_+$ also known as SHIQ. The DL reasoned FaCT++ implements a tableax decision procedure for the well known SHOIQ description logic, with additional support for datatypes, including strings and integers. The system employs a wide range of performance enhancing optimizations, including both standard techniques and newly developed ones.

Recently, some researchers have proposed some methods to do the ontology reasoning tasks, article [17] provides a rigorous treatment of data type predicates on the concrete domain. It investigated the complexity of combined reasoning with description logics and concrete domains, and extended ALL (D), which is the basic description logic for reasoning with concrete domain, by the operators “feature agreement” and “feature disagreement”. Jeff Z. Pan [18] proposed a flexible reasoning architecture for Semantic Web ontology languages and described the prototype implementation of the reasoning architecture, based on the well-known FaCT DL reasoned. It allows users to define their own data types and data type predicates based on built-in ones and new data type reasoners can be added into the architecture without having to change the concept reasoned. Article [4] present OWL Flight, which is loosely based on OWL, but the semantics is grounded in Logic Programming rather than Description Logics, and it borrows the constraint-based modeling style common in databases. And the ontology reasoning tasks are supported by OWL DL and OWL Flight.

Similarly to our approach, article [9] proposed a new novel application domain for Alloy firstly. It believed that software engineering techniques and tools can provide automatic reasoning and consistency checking services for Semantic Web. The software modeling languages Z and its proof tool Z/EVES can also be used to verify DAML+OIL [8]. And in the article [7], a combined approach is proposed to checking Web ontologies. It used the software engineering techniques and tools, i.e., Z/EVES and Alloy Analyzer, to complement the ontology tools for checking Semantic Web documents.

We take advantage of Alloy and Alloy Analyzer to convert the OWL ontologies into Alloy model, and then using the Alloy Analyzer to automatically check and reason the generated model.

Compared with the article[23], our method can handle complex-property ontology efficiently. In the article[23], the ontology classes and properties are translated into different signatures in Alloy model, and then predicates are used to establish the relationship between them. Our approach uses the features of Alloy to convert the ontology properties to relations of Alloy model directly, for in Alloy everything is a relation. For the Alloy model, there are striking contrasts between the two methods with different scopes. In the Figure 8, there is a big difference in numbers in the analysis of the same ontology. The horizontal axis represents the Alloy model scope, while the vertical axis represents the number of variables. You can see that the greater the scope given is，the bigger the difference of the number of variables is, which is to say

that the more the number of ontology instances is, the more obvious the advantage of this method is.

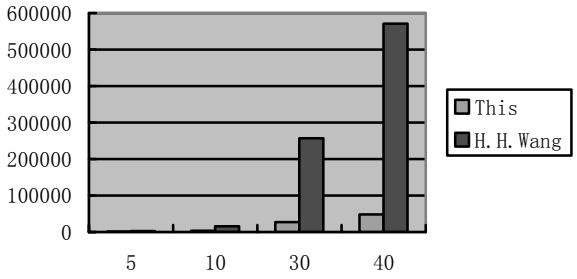


Figure 8. The variables of a same model in different methods.

The Figure 9 is similar to the previous figure, the horizontal axis represents the Alloy model scope, while the vertical axis represents the number of clauses.

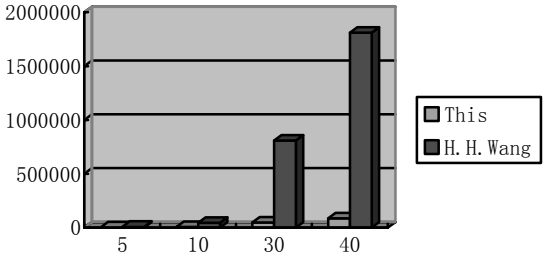


Figure 9. The clauses of a same model in different methods.

The Figure 10 is to illustrate the difference in execution time (ms).

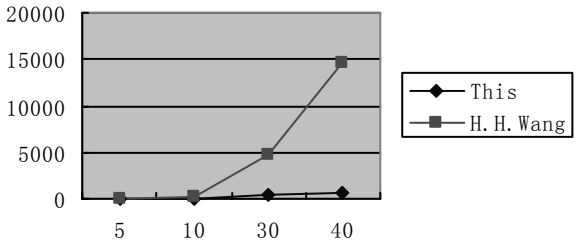


Figure 10. The time of a same model in different methods.

V. Conclusion

This paper presented a reasoning method for OWL ontologies. Compared to the existing reasoners for OWL, we propose a method using software engineering tools. The paper is based on the conversion mode. It can meet the complex-properties ontology reasoning. And better than the existing reasoners, it can give reasoning services in the instance level. Further, the reasoning of Alloy Analyzer can not only prove the ontology is wrong, generate a set of ontology instances, counterexample on predicates, but also can offer help to point out where the errors are.

However, this method also has its shortcomings, it can't be carried out automatically, it requires human involvement.

VI. Acknowledgments

This work is supported by the National Natural Science Foundation of China (61175056), Young Key Teachers Foundation Projects of Dalian Maritime University (2011QN028) and the Open Project Program of Artificial Intelligence Key Laboratory of Sichuan Province, (Sichuan University of Science and Engineering), China (2011RYJ03).

References

[1] B. FRANZ, C. DIEGO, L.M. DEBORAH, N. DANIELE AND F.P.-S. PETER Eds. *The description logic handbook: theory, implementation, and applications*, Cambridge University Press, 545. 2003.

[2] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. "DAML-S: Semantic markup for Web services". In Proceedings of the International Semantic Web Working Symposium (SWWS), pages 411-430, 2001.

[3] T. Berners-Lee, J. A. Hendler and O. Lassila, "The Semantic Web". Scientific American. v284 iMay (5). 34-43.

[4] J. De Bruijn, R. Lara, A.Polleres, and D. Fensel, "OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic Web". In Proceedings of the Proceedings of the 14th international conference on World Wide Web, Chiba, Japan 2005 ACM, 1060836, 623-632.

[5] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seavorne AND K. Wilknson, "Jena: implementing the semantic web recommendations". In Proceedings of the Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters, New York, NY, USA2004 ACM, 1013381, 74-83.

[6] 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+OIL". IEEE Intelligent Systems 17, 40-47.

[7] J. S. Dong, C. H. Lee, H. B. Lee, Y. F. Li and H. Wang. "A combined approach to checking web ontologies". In Proceedings of the Proceedings of the 13th international conference on World Wide Web, New York, NY, USA 2004 ACM, 988770, 714-722.

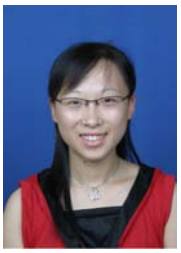
[8] J. S. Dong, C. H. Lee, H. B. Lee, Y. F. Li and H. "Verifying DAML+OIL and Beyond in Z/EVES". In Proceedings of the Proceedings of the 26th International Conference on Software Engineering2004 IEEE Computer Society, 999425, 201-210.

[9] J. S. Dong, J. Sun and H. Wang. "Checking and reasoning about Semantic Web through alloy". In Fme 2003: Formal Methods, Proceedings, K. Araki, S. Gnesi and D. Mandrioli Eds. Springer-Verlag Berlin, Berlin, 796-813.

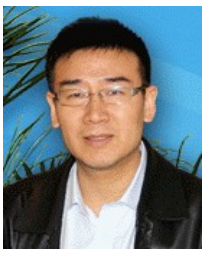
[10] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann and M. Klein. "OIL in a nutshell". In Knowledge Engineering and Knowledge Management, Proceedings - Methods, Models, and Tools, R. DIENG AND O. CORBY Eds. Springer-Verlag Berlin, Berlin, 1-16. 2000.

[11] T. R.Fruber. "Toward principles for the design of ontologies used for knowledge sharing". Int. J. Hum.-Comput. Stud. 43, 907-928. 1995.

[12] I. Horrocks, “The FaCT System” In Proceedings of the Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, vol. LNAI 1397, pp. 307-312. 1998.
[13] I. Horrocks, D. Fensel, J. Boriekstra, S. Decker, M. Erdmann, C.Goble et al. The Ontology Inference Layer OIL, Tech. Report, Vrije Universiteit, Amsterdam. Retrieved March 19, 2002.
[14] I. Horrocksm, P. F. Patel-Schneider, S. Bechhofer and D. Tsarkov.”OWL rules: A proposal and prototype implementation”. Journal of Web Semantics 3, 23-40. 2005.
[15] D. Jackson. “Alloy: a lightweight object modelling notation”. ACM Trans. Softw. Eng. Methodol. 11, 256-290. 2002.
[16] D. Jackson, I. Schechter and H. Shlyahter. “Alcoa: the alloy constraint analyzer”. In Proceedings of the Proceedings of the 22nd international conference on Software engineering, Limerick, Ireland2000 ACM, 337616, 730-733. 2000.
[17] C. Lutz. “The Complexity of Reasoning with Concrete Domains Revised Version”. Rheinisci-Westfalische Technische Hochschule. 1999.
[18] J. Z. Pan. “A flexible ontology reasoning architecture for the Semantic Web”. Ieee Transactions on Knowledge and Data Engineering 19, 246-260. 2007.
[19] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur and Y. Katz. “Pellet: A practical OWL-DL reasoned”. Journal of Web Semantics 5, 51-53. 2007.
[20] Y. Song and R. Chen. “Non-standard Reasoning Services for the Verification of DAML+OIL Ontologies”. In Artificial Intelligence Applications and Innovations, H. PAPADOPOULOS, A. ANDREOU AND M. BRAMER Eds. Springer Boston, 203-210-210. 2010.
[21] D. Tsarkov and I. Horrocks. “FaCT++ Description Logic reasoner: System description”. In Automated Reasoning, Proceedings, U. FURBACH AND N. SHANKAR Eds. Springer-Verlag Berlin, Berlin, 292-297. 2006.
[22] H. Volker, M. Ralf and A. Turhan. “RACER User"s Guide and Reference Manual Universitaet Hamburg”. 1999.
[23] H. Wang, J. Dong, and J. Sun. “Reasoning support for Semantic Web ontology family languages using Alloy”. Multiagent and Grid Systems 2. 2006.
[24] A. Alnusair and T. Zhao. “Using Ontology Reasoning for Reverse Engineering Design Patterns”. Models in Software Engineering, 344–358. 2010.
[25] R. Umeton, B. Yankama, G. Nicosia and C.F. Dewey. “OREMP : Ontology Reasoning Engine for Molecular Pathways Use Case 1 : Model Alignment for Parallel Simulation with Cytosolve Use Case 2 : Aligned Ontologies”. Lloydia Cincinnati, 2122-2122. 2009.
[26] I. Conference and C. Academy. “Decomposition Model Building and Ontology Reasoning Towards G-Layer of RGPS Requirement Meta-Model”. Computer, 47-51. 2010.



Yingjie Song born in ShanDong province, China, in September 1983. She earned master degree of Computer Science and Technology from the school of information science and technology at the Dalian Maritime University, Dalian, China in 2009, and now study in the school of information science and technology at the Dalian Maritime Univerisity as a Doctoral Candidate. His research interests are in Semantic Web, Ontology Evolution and Ontology reasoning, etc.



Rong Chen received the Ph.D. degree in computer software and theory from the Jilin University, China, in 2000. He is currently a member of the school of information science and technology at the Dalian Maritime University, China. He has previously held position at Sun Yat-sen University. His research interests are in system reliability and automated software engineering. joined the faculty in 2005 having previously worked at the Sun Yat-sen University and the Graz University of Technology. Currently, he research focuses on trusted software, Internet and mobile computing, semantic web, embedded software engineering, database and knowledge base, and intelligent diagnosis.

Yaqing Liu is a lecturer in the School of Information Science and Technology at Dalian Maritime University, Dalian, China. He received PhD degrees in Computer Application Technology in 2011. His research interests are in Semantic Web, Ontology Evolution and Ontology reasoning, etc. He has authored over 10+ refereed journal/conference papers and about 80% papers is indexed by SCI/EI.