Non-standard Reasoning Services for the Verification of DAML+OIL Ontologies

Yingjie Song, Rong Chen

Non-Standard Reasoning Services for the Verification of DAML+OIL Ontologies

Yingjie Song, Rong Chen

School of Information Science & Technology.
Dalian Maritime University, Dalian 116026, P. R. China
{tiantian_yingjie@sina.com, tsmc.dmu@gmail.com}

Abstract. Ontology has a pivot role in the development of Semantic Web which provides the understanding of various domains that can be communicated between people and applications. Motivated by J. S. Dong's work, we propose a new approach to interpreting DAML+OIL in a lightweight modeling language for software design, Alloy, which is used to provide a non-standard reasoning service for the verification of DAML+OIL ontologies. To do so, Jena is first used to parse ontology documents into classes, properties and statements, next we use algorithms to translate them into Alloy model, the Alloy Analyzer is then used to check and reason about such model. The experiments show that our method greatly improves J. S. Dong's work, and distinguishes from the traditional ontology reasoners in property checking and reasoning.

Keywords: Ontology Reasoning; DAML+OIL; Alloy; Semantic Web

1 Introduction

A Semantic Web [4], as the next generation of the Web, provides well-defined notations and techniques for humans and applications to quickly and accurately access Web information and services. In the development of Semantic Web there is a pivot role of ontology, since ontology languages provide modeling primitives for converting notations in nature language into machine-readable logical formulas, from which autonomous software agents may infer and come to conclusions [1].

Ontology languages, such as DAML+OIL, enhance computer programs through structured organizational information and rules, with which it is able to understand the logical relationship between them. Ontology reasoning is crucial in that inconsistent ontology cannot be shared or used by autonomous software agents. A number of ontology inference engines, such as FaCT [3], RACER [2], and FaCT++ [10] have been developed with the advancement of ontology languages to facilitate ontology creation, management, verification, merging, etc.. However, the checking and reasoning of complex ontology-related properties cannot be done by them.

There is a role for software engineering techniques and tools that contribute to the Semantic Web development. J. S. Dong first proposes the use of Alloy [5] in checking and reasoning about the semantic relationship between web resources [1]. We propose a novel transformation from DAML+OIL ontology to Alloy, which greatly improves J. S. Dong's work; our approach scales up well and can work on a larger scope of property checking.

The rest of the paper is organized as follows: Section 2 gives a brief introduction to DAML+OIL and Alloy. In the section 3, a simple ontology example is given described in DAML+OIL, and then the ontology document is analyzed by jena, the results of which will be used as the inputs of the algorithms, which are used to transferred the DAML+OIL into Alloy model.

2 Overview of DAML+OIL and Alloy

2.1 Logical Characteristic of DAML+OIL

DAML+OIL [7] is a successor language to DAML [8] and OIL [9] that builds on earlier W3C standards such as RDF, RDF Schema, and the language components of OIL. DAML+OIL layered on top of RDFS it inherited RDFS ontological primitives (subclass, range, domain). As a semantic Web ontology language, DAML+OIL provides users a richer set of modelling primitives (transitivity, cardinality, …) that are commonly found in frame-based languages.

Although DAML+OIL is tightly integrated with RDFS, which provides the only specification of the language and its only serialization, DAML+OIL defines the semantic of the language to give a meaning to any ontologies that conform to the RDFS specification, including "strange" constructs such as slot constraints with multiple slots and classes. It contains richer modeling primitives than RDF. This is made easier by the fact that the semantics of DAML+OIL is directly defined in both a model-theoretic and an axiomatic form. Theoretically, DAML+OIL is undecidable, but its processor to detect the occurrence of constraints and warn the user of the consequences.

2.2 Alloy

Alloy [5] is a textual, declarative modelling language rooted in first order relational logic, which is widely accepted as micromodels of software in the software engineering community. For relationships between web resources are focus point in the Semantic Web, we believe that it will be a new application domain for Alloy. 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 comes with a tool, the Alloy Analyzer [6], which supports fully automated analysis of Alloy models through simulation and *Assertion* checking. While *Assertions* are assumptions about the model that can be checked. Simulation yields a random instance that is consistent with the model. 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.

3 Description of the Approach

The specific process of our approach on ontology reasoning is shown in Fig.1: ontology documents are analyzed by Jena, and the results contain three parts: Classes C, Property P and Statements S. Next, the results are converted into Alloy model using daml2Alloy algorithm. And then we use Alloy Analyzer to check the model. In case an error, we check back the original ontology and correct it accordingly. The jena is used again to check the corrected ontology. These three steps are explained in more detail in the following.

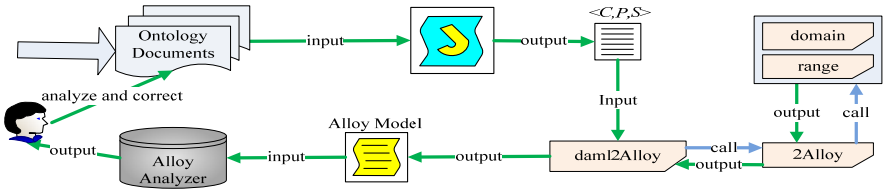

Fig.1. The specific process of ontology reasoning.

3.1 Parsing DAML+OIL Documents

To handle DAML+OIL ontologies, we adopt Jena [10] as a frontend of our framework to parse textual DAML+OIL documents. Jena provides APIs for maniplulating RDF graphs, abstracting from which it provides the ontology API for OWL and DAML ontologies. Our DAML+OIL parser, based on Jena parser, reads a DAML+OIL document of animal ontology (shown in Fig.2), which defined four classes: `Animal`, `Male`, `Man` and `Female`. While `Man` is subclass of `Male`, `Male` and `Female` are disjointed subclasses of `Animal`. `hasFather`, `hasParent` and `hasChild` are three properties such that `hasParent` and `hasChild` are inverse to each other and `hasFather` is subproperty of `hasParent`. We translate it into RDF triples, which are composed of *Classes*, *Properties* and *Statements*.

```
.....
<rdfs:Class rdf:about="Animal">
  <rdfs:label>Animal</rdfs:label>
  <rdfs:comment>
    This class of animals is
illustrative of a number of ontological
idioms.
  </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:about="Male">
  <rdfs:subClassOf
rdf:resource="Animal"/>
</rdfs:Class>
<rdfs:Class rdf:about="Female">
  <rdfs:subClassOf
rdf:resource="Animal"/>
  <daml:disjointWith
rdf:resource="Male"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:about="Man">
  <rdfs:subClassOf rdf:resource="Male"/>
</rdfs:Class>
.....
<rdf:Property rdf:about=" hasParent">
  <rdfs:domain rdf:resource=" Animal"/>
  <rdfs:range rdf:resource=" Animal"/>
</rdf:Property>
<rdf:Property rdf:about=" hasFather">
  <rdfs:subPropertyOf      rdf:resource="
hasParent"/>
 <rdfs:range rdf:resource=" Male"/>
</rdf:Property>
<rdf:Property rdf:about=" hasChild">
<daml:inverseOf           rdf:resource="
hasParent"/>
</rdf:Property>
   .....
```

Fig.2. A DAML+OIL document of animal ontology

We use a simple ontology about animal as a running example to show the outputs of Jena. It contains a sequence of *Classes*, *Properties* and *Statements*, each having a couterpart in the original document. Such *Classes*, *Properties* and *Statements* provide programmatic objects like DAMLClass, DAMLProperty and RDFTriples for our coversion algorithm in the next section. As illustrated in Fig. 1, our DAML+OIL parser reads such a textual document and converts it into RFD triples as follows:

```
<DAMLClass Animal>
<DAMLClass Male>
<DAMLClass Man>
<DAMLClass Female>
The number of Classes is: 4
<DAMLProperty hasFather>
<DAMLProperty hasParent>
<DAMLProperty 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 Statemenst is: 9.
```

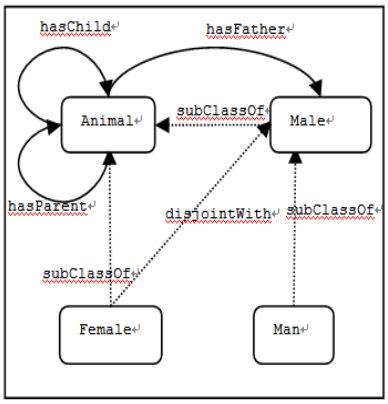

Fig. 3. Jena outputs of "Animal" ontology

3.2 Generation of the Alloy Model

Given RDFTriples and the related classes, we move on to generate the equivalent Alloy model, which contains Alloy classes and the relationship between them.

As shown below, our Algorithm daml2Alloy (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. To do so, we map each class into an Alloy signature, and the relationships between such classes are represented by Alloy primitives such as *Relations*, *Facts*, *Functions* and *Predicates*.

Algorithm 1: daml2Alloy (C, P, S, Σ)
Input: a set C of *Classes*, a set P of *Properties* and a set S of *Statements*
Output: a textual Alloy model Σ

1. $\Sigma \leftarrow \omega$;
2. **FOR** each $c \in C$
3. $\Sigma \leftarrow \Sigma$ + 2Alloy(c, S);
4. **FOR** each $s \in S$
5. **IF** *s.predicate.name* = "disjointWith"
6. **IF** *s.subject.subClassOf* ≠ *s.object.subClassOf*
7. $\Sigma \leftarrow \Sigma$ + "pred {no c1:" + *s.subjec.name* +", c_2:" + *s.object.name* +" | $c_1 = c_2$}";
8. **IF** *s.predicate.name* ="complementOf"
9. $\Sigma \leftarrow \Sigma$ + "pred {" + *s.subject.name* +"=" + C – *s.object.name* +"}";
10. **IF** *s.predicate* meets one of Conditions in Table 3
11. the corresponding generated Alloy predicate is appended to Σ
12. **RETURN** Σ;

In our algorithm, the Alloy model Σ is initialized to be an empty string ω on line 1, each class c is converted into a signature by invoking 2Alloy (c, S) algorithm (see below), next follows the conversion of each statement into *Predicates* in a loop through lines 4~11, finally the produced model Σ is returned. Since each statement

$s \in S$ contains RDF elements in the form of a RDF triple <*subject*, *predicate*, *object*>, we use *s.subject*, *s.predicate*, *s.object* to denote the three RDF elements respectively. Moreover, we further use *e.name* (*e.subClassOf*) to represent the name of a RDF element *e* (its parent's class) in that each RDF triple depicts the relationship between RDF elements. For instance, when it comes to two disjointed classes on lines 5~6, we should further consider whether they have the same parent class, if not, a new Alloy predicate is generated for depicting such a constraint[1]. Lines 8~9 handle a new case like lines 5~6, more generation on lines 10~11 is summarized in cases in Table 3, where a specific textual Alloy predicate is generated when some condition holds.

Table 3 More cases for converting statements to Alloy

Case	Condition	Alloy predicates generated
1	*s.predicate.name*="subPropertyOf"	"pred subPropertyOf{all r:"+ *s.subject.range* +"| r in "+ *s.object.range* +"}"
2	*s.predicate.name*="samePropertyAs"	"pred samePropertyAs{"+*s.subject*+ "=" +*s.object*+"}"
3	*s.predicate.name*="inverseOf"	"pred inverseOf{"+*s.subject*+"=~"+*s.object*+"}"
4	*s.predicate.name*="TansitiveProperty"	"pred TansitivePropertyOf { a,b,c ∈" +*s.subject*+" | a.("+*s.predicate*+") = b && b.("+*s.predicate*+") =c ⇒ a. ("+ *s.predicate*+") = c}"
5	*s.predicate.name*="UniqueProperty"	"pred UniqueProperty{#("+*s.predicate.range*+")=1}"
6	*s.predicate.name*="UnambiguousProperty"	"pred UnambiguousProperty { # ("+ *s.predicate.domain* +") =1 }"
7	*s.predicate.name*="toClass"	"pred toClass{all (("+*s.predicate.domain* +").("+*s.predicate*+")) in ("+*s.predicate.range*+")}"
8	*s.predicate.name*=hasClass	"pred hasClass{ some((" + *s.predicate.domain* +").("+*s.predicate*+")) in ("+*s.predicate.range*+")}"
9	*s.predicate.name*=hasValue	"pred hasValue{#("+*s.predicate.range*+")=1}"
10	*s.predicate.name*=cardinality	"pred cardinality{#("+*s.predicate.range*+")=" + *s.object*"}"
11	*s.predicate.name*=maxCardinality	"pred maxCardinality{#("+*s.predicate.range*+")<= "+*s.object* "}"
12	*s.predicate.name*=minCardinality	"pred minCardinality{#("+ *s.predicate.range*+")>="+*s.object* "}"
13	*s.predicate.name*=hasClassQ	"pred hasClassQ{ some(("+*s.predicate.domain*+").("+*s.predicate*+")) in ("+*s.predicate.range*+")}"
14	*s.predicate.name*=cardinalityQ	"pred cardinalityQ{#("+*s.predicate.range*+")="+*s.object*+"}"
15	*s.predicate.name*=maxCardinalityQ	"pred maxCardinalityQ{#("+*s.predicate.range*+")<="+ *s.object*+" }"
16	*s.predicate.name*=minCardinalityQ	"pred minCardinalityQ{#("+*s.predicate.range*+")>="+*s.object* "}"

The next algorithm 2Alloy (*c*, *S*) is used to produce a signature for a class *c* with respect to a set *S* of *Statements*. The idea behind this conversion is as follows: let *c* be the input class, we first create an Alloy signature named as *c.name* on line 1. If *c* has parent class, i.e., *c.subClassOf* is not empty by checking the input *Statements*, we think *c* extends its parent class *c.subClassOf.name* on line 3. When it comes to a property on line 5, its domain and range is calculated before appending the resulting signature σ on line 7.

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

As shown in Algorithm 3, the domain of a property is calculated recursively; the domain is associated with an object on line 2 when a RDF triple satisfies such a condition that its subject is a property and its predicate depicts a domain. Otherwise, the parent property is recursively checked until a qualified RDF triple is reached. We

[1] No constraint is generated when two disjointed classes have the same parent, because Alloy 4.0 defaultly assumes classes are disjointed.

use *parent (property)* to represent the parent property of the parameter *property*. Note that the calculation of the range of a property with respect to *Statements* is quite similar to Algorithm 3, we omit it for obviousness.

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

Taking the "Animal" ontology as an example again, it is used to show how the conversion is achieved. The DAML+OIL document will be transferred into Alloy model as Fig.4.

```
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}
```

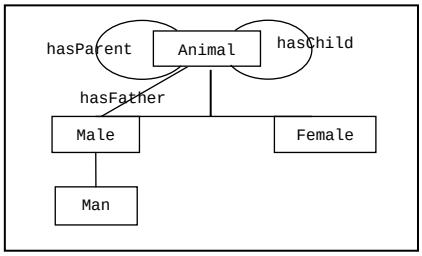

Fig. 4. The generated Alloy model of "Animal" ontology

3.3 Verifying Ontologies with the Alloy Analyzer

Semantic Web reasoning is one of key issues for ontology design, construction and maintenance, which contains the ontology consistency checking, subsumption reasoning, which task is to derive a class is another's parent, and implication relation checking. The correct ontology required to meet at least on instance. This is achieved through Alloy Analyzer to generate an instance of the model in given scope. As shown in Figure 5, there is an inconsistency occurred in the "Animal" model. This is because there is a `subProperty` constraint between `femalHasFather` and `animalHasFather`, `Woman` is `subClassOf Male` is implied since `Male` is the range of `animalHasFather` and `Woman` is the range of `femaleHasFather`.

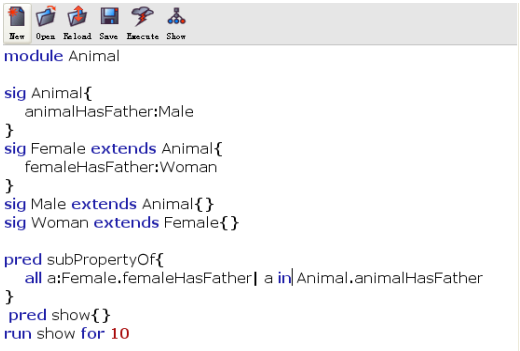

Fig.5. Checking result of Alloy Analyzer

Note that if Alloy Analyzer can't find a solution, it maybe for the too small scope. If there is something of inconsistent in reasoning with Alloy, assertion is an important criterion. When Alloy Analyzer can't find a counterexample, the assertion is reasonable, else Alloy Analyzer will generate a counterexample and some predicates and facts should be added to improve the model.

4 Comparison of Experimental Results

In J. S. Dong's article, classes and properties in semantic ontology were converted to subclasses of resource, and then predicates were used to establish the relationship between different resources. As a proof tool of program correctness, theorem machine prove and knowledge representation, the shortcoming of first order logic is that in reasoning it prone to "combinatorial explosion". Method we used is to make use of the characteristics of Alloy, we convert the properties of the semantic ontology to the properties of the signature in Alloy model. It has been greatly improved in the scale. Table 4 shows the operation results in the case of the same scope of the two methods.

Table 4 Comparison of Experimental Data

	scope	GenerateCM + GeneratePM				J. S. Dong			
		vars	primary vars	clauses	time	vars	primary vars	clauses	time
Family	(5)	1074	135	1674	32	3025	265	7875	47
	(10)	3168	520	5281	63	15530	1430	44475	156
	(30)	27468	4560	48201	391	257182	30090	806831	4828
	(40)	48618	8080	85861	578	570992	69320	1810146	14688
Course	(10)	5477	440	11868	63	11845	1440	31357	125
	(20)	17507	1680	41626	203	67632	9480	197133	859
	(30)	38737	3720	93696	453	204687	30120	612883	3578
	(40)	69887	6560	171406	813	460162	69360	1396873	8812
airportCode	(5)	1196	125	1910	16	2416	270	5573	31
	(20)	14716	1700	26300	156	67758	9480	193719	1109
	(35)	44656	5075	81200	688	44656	5075	81200	5781
	(50)	90796	10250	166250	1422	866928	13200	2627124	17422
Document	(10)	11458	1800	19691	125	31271	2070	87236	375
	(20)	37528	6000	67371	422	169783	10740	535897	2172
	(30)	80318	12600	146711	1125	501543	32010	1655392	6860
	(40)	139108	21600	256451	2235	1109903	71880	3750587	31016

As is shown in Table 4, we analyze and compare with several ontologies. Variables, primary variables, clauses and runtime are the four mainly elements for comparison. For each ontologies, we give four scopes to illustrate that with the expansion of the scope, our approach reflects the increasingly better performance comparing to the method of article [1].

The first ontology we used is the Family ontology, which contains three classes and five properties. Its Alloy model is composed of three signatures, five properties and one predicate. As is shown in Table 4, with the scope increasing, the advantages are more and more obvious. When the scope is equal to 40, the number of variables is 10 times larger than ours, the number of primary variables is more than 8 times and the runtime is 25 times. It is the similar with the other examples.

5 Conclusion

We propose an approach to convert the DAML+OIL ontology to Alloy model, and then using the Alloy Analyzer to automatically check and reason the generated model. First, we use Jena to analyze the ontology document and get classes, properties and statements of it. Next, we propose two algorithms to generate the Alloy model of the ontology. Finally, the tasks of checking and reasoning are executed by Alloy Analyzer. We have applied our approach to several ontologies, it can discover errors and inconsistencies, and when there are something wrong, we can efficiently correct the errors on the assistance of the counterexample given by Alloy Analyzer.

6 Acknowledgments

This work is supported by National Natural Science Foundation of China (60775028), the Major Projects of Technology Bureau of Dalian No.2007A14GXD42, and IT Industry Development of Jilin Province.

References

1. Hai Wang Jin Song Dong, Jing Sun. Checking and reasoning about semantic web through alloy. In Proceedings of Formal Methods Europe: FME'03, 2805 of Lect.:796--814, 2003.
2. Volker Haarslev and Ralf Möller. RACER User's Guide and Reference Manaual: Version 1.7.6, Dec 2002.
3. Ian Horrocks. The fact system. In TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, pages 307--312, London, UK, 1998. Springer-Verlag.
4. J.Hendler T.Berners-Lee and O.Lassila. The semantic web. Scientific American, May 2001.
5. Daniel Jackson. Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol., 11 (2):256--290, 2002.
6. Daniel Jackson, Ian Schechter, and Hya Shlyahter. Alcoa: the alloy constraint analyzer. In ICSE '00: Proceedings of the 22nd international conference on Software engineering, pages 730--733, New York, NY, USA, 2000. ACM.
7. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. Reviewing the design of daml+oil: an ontology language for the semantic web. In Eighteenth national conference on Artificial intelligence, pages 792--797, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
8. F.van Harmelen, P. F. Patel-Schneider, and I. Horrocks (editors). Reference description of the daml+oil ontology markup language. March, 2001.
9. Jeen Broekstra, Michel Klein, Stefan Decker, Dieter Fensel, and Ian Horrocks. Adding formal semantics to the web building on top of rdf schema. In In Proc. of the ECDL 2000 Workshop on the Semantic Web, 2000.
10. Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006, pages 292--297. Springer, 2006.
11. Lutz C. The complexity of reasoning with concrete domains revised version. Technical report, 1999.