Reasoning Support for Semantic Web Ontology Family Languages Using Alloy

Hai H. Wang
Department of Computer Science
The University of Manchester, United Kingdom
hai.wang@cs.manchester.ac.uk

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

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

Jun Sun
School of Computing
National University of Singapore
sunj@comp.nus.edu.sg

August 14, 2006

Abstract

Semantic Web (SW), commonly regarded as the next generation of the Web, is an emerging vision of the new Web from the Knowledge Representation and the Web communities. To realize this vision, a series of techniques has been proposed. Semantic Web Ontology Langauge (OWL) and its extension Semantic Web rule Language (SWRL) and Semantic Web Logic Language (SWRL-FOL) are some of the most important outputs from the SW activities. However the existing reasoning and consistency checking tools for those languages are primitive. This paper proposes using the existing formal modelling tool, in particular Alloy, to provide an automatic reasoning service for the Semantic Web ontology family languages (OWL/SWRL/SWRL-FOL). [1]

keywords: Semantic Web, Alloy, OWL, SWRL, FOL.

[1]Primary contact: Hai H. Wang, School of Computer Science, The University of Manchester, M13 9JPL, United Kingdom Email: hai.wang@cs.manchester.ac.uk Telephone: +44 161 275 0686 Fax: +44 161 275 6204

1 Introduction

The power of the Semantic Web [1], as the next generation of the Web, will be realized when software agents are able to understand the Web content, process the information and exchange the results with other software agents. Adding logic to the Web is one of the key requirements. This logic must be powerful enough to describe complex properties of web resources but not so complicated that agents could be tricked by being asked to consider a paradox. To achieve these two contradictory requirements, researchers attempt to adopt the layered approach, where the upper layer is extended from the lower layer with enhanced expressive power. This allows that different applications can choose the logic language suiting their needs most.

The bottom layer is Web Ontology Language (OWL) [14]. OWL, a recommendation by the World Wide Web Consortium (W3C), is the latest standard to define the ontology. It is based on Description Logic (DL). Although OWL adds considerable expressive power to the Semantic Web, to retain the decidability of key inference problems in OWL DL and OWL Lite, OWL has its expressive limitations. Certain desired properties can not be expressed for some applications. Semantic Web Rule Language (SWRL) [9] extends OWL by combining the OWL DL and OWL Lite with the Unary/Binary Catalog sub-languages of the Rule Markup Language. It introduces a new kind of axiom, named Horn clause rules, to OWL DL. Recently, Semantic Web Rule Language First Order Logic (SWRL-FOL) [15] has been proposed to further extend the SWRL to handle unary/binary first-oder logic.

Reasoning can be useful at many stages during the design, maintenance and deployment of ontology. Using the reasoning service provided by Semantic Web reasoners, software agents can autonomously infer new knowledges from the given knowledges and perform different tasks. For example, Pizzafinder [2], as a small application developed by our research group, demonstrates how the reasoner could be used by a web software agent for their application. Pizzafinder uses the Pizza Ontology, and a reasoner to dynamically generate pizza toppings and pizza topping categories. The user can include toppings that they would like on their pizza and exclude any toppings that they do not want on their pizza. The description logic reasoner is used to determine if the choices that have been made contradict each other - for example, choosing to include Jalapeno Pepper topping, but choosing to exclude all hot toppings - the choices are automatically adjusted to modify any decisions that could potentially lead to contradictions and inconsistent results.

Because autonomous software agents may perform their reasoning and come to conclusions without human supervision, it is essential that the shared ontology is consistent. However, since the Semantic Web technology is still in the early stage, the reasoning and consistency checking tools are primitive. The existing OWL reasoning tools such as FaCT [8] and RACER [6] have been developed specifically for the decidable description logic, which are based on tableaux algorithm. They are far from perfect. Furthermore, currently there does not exist a tableaux algorithm that can support the reasoning of SWRL-FOL, or even SWRL. Hence, it would take some effort and time for people to research into new algorithms and build new tools to support

[2] http://www.co-ode.org/downloads/pizzafinder/

SWRL and SWRL-FOL reasoning. However, as it can been foreseen that it is critical and urgent to provide some reasoning service to SWRL and SWRL-FOL in order to make them to be integrated into ontology languages hierarchy and to have their impacts on the practical web applications.

Alternatively, rather than developing new algorithms and tools, a light-weight approach to provide reasoning service which can complement existing OWL reasoners and support SWRL and SWRL-FOL is to customize and reuse some existing tools. After decades of research and development, some mature formal modelling/reasoning tools have been established successfully. These tools could well be adopted to reasoning about OWL, SWRL and SWRL-FOL.

This paper proposes to develop a reasoning environment using the software modelling language Alloy and its Analyzer [11] for web ontology families language. It complements the existing OWL reasoning tools like RACER, and also supports the newly extended SWRL and SWRL-FOL.

The rest of the paper is organized as follows. Section 2 briefly introduces the OWL, SWRL, SWRL-FOL and Alloy. In section 3, we present the Alloy semantics for the OWL/SWRL/SWRL-FOL language and the transformation from the ontologies into their corresponding Alloy models. Section 4 presents a case study to demonstrate the reasoning processes of SWRL-FOL ontology models in the Alloy Analyzer. Section 5 concludes the paper and discusses the future work.

This paper is substantially extended and revised from the early conference paper 'Reasoning Support for SWRL-FOL Using Alloy' [19].

2 Backgrounds

2.1 Semantic web overview

2.1.1 Semantic web and OWL

The Semantic Web is a vision for a new kind of Web with enhanced functionality which will require semantic-based representation and processing of Web information. W3C has proposed a series of technologies that can be applied to achieve this vision. The Semantic Web extends the current Web by giving the web content a well-defined meaning, better enabling computers and people to work in cooperation. XML is aimed at delivering data to systems that can understand and interpret the information. XML is focused on the syntax (defined by the XML schema or DTD) of a document and it provides essentially a mechanism to declare and use simple data structures. However there is no way for a program to actually understand the knowledge contained in the XML documents.

Resource Description Framework (RDF) [12] is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF uses XML to exchange descriptions of Web resources and emphasizes facilities to enable automated processing. The RDF descriptions provide a simple ontology system to support the exchange of knowledge and semantic information on the Web. RDF Schema [2] provides the basic vocabulary to describe RDF documents. RDF Schema can be used to define properties and types of the web

OWL constructs	Description
OWL_class	classes
$OWL\_property$	properties
$OWL\_subclass[C]$	subclasses of C
$OWL\_subproperty[P]$	subproperties of P
$instanceof[C]$	instances of the OWL class C

Table 1: OWL constructs (partial)

resources. The advent of RDF Schema represented an early attempt at an SW ontology language based on RDF.

OWL [14] is a standard (W3C recommendation) for expressing ontologies in the Semantic Web. The OWL language facilitates greater machine understandability of Web resources than that supported by RDFS by providing additional constructors for building class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. OWL consists of three sub-languages: OWL Lite, OWL DL and OWL Full, with increasing expressiveness. OWL Lite and DL are decidable, but OWL Full is generally not. An OWL ontology consists of classes, properties and individuals. Classes are interpreted as sets of objects that represent the individuals in the domain of discourse. Properties are binary relations that link individuals, and are interpreted as sets of tuples, which are the subsets of the cross product of the objects in the domain of discourse. OWL classes fall into two main categories – named classes and anonymous classes. Anonymous classes are formed from logical statements. They contain the individuals that satisfy the logical description. Anonymous classes may be further sub-divided into restrictions and logical class expressions. We summarize some essential OWL constructs in Table 1. To be simpler, informal OWL syntax has been used here. For example, OWL_class denotes the OWL construct "Class" .

2.1.2 SWRL

Although OWL includes a relatively rich set of class constructors, the language provided for expressing properties is much weaker. SWRL [9] intends to overcome the expressive restriction of OWL properties by extending OWL with some form of "rule language". SWRL is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Catalog sublanguages of the Rule Markup Language. SWRL introduces a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sub-languages of OWL. SWRL extends OWL by also allowing rule axioms, i.e., by adding the construct:

$$axiom ::= rule$$

A rule axiom consists of an antecedent and a consequent, each of which consists of a set of atoms which could be class membership (*C(x)*), property membership (*P(x,y)*) or individual in/equalities(*differentFrom(x,y)/sameAs(x, y)*). Informally, a rule means that if the antecedent holds (is "true"), then the consequent must also hold. A simple

example of the rules could be used to express the knowledge that "if $?x1$ is a child of $?x2$ and $?x2$ is a brother of $?x3$, then $?x3$ is an uncle of $?x1$". Informally, this rule could be written as:

$$hasChild(?x2, ?x1) \wedge hasBrother(?x2, ?x3) \\ \Rightarrow hasUncle(?x1, ?x3)$$

2.1.3 SWRL-FOL

SWRL-FOL [15] extends SWRL axiom to arbitrary first-order formula over unary and binary predicates. It extends SWRL with "assertion" axioms that contain first-order sentences , i.e.

$$axiom ::= assertion$$

Assertions assert first-order sentences, where no free variables are allowed in the formulae. For example an axiom could be used to express the knowledge that "for all the person $?x1$ if s/he is a 'wealthy Parent', then s/he has at least one child $?x2$ who is a millionnaire.". Informally, this axiom could be written as:

$$\forall ?x1 \mid wealthyParent(?x1) \Rightarrow \\ \exists ?x2 \mid hasChild(?x1, ?x2) \wedge millionnaire(?x2)$$

We can see from the above example that by introducing first-order formulas, more complex logical statements can be expressed in SWRL.

2.1.4 Existing reasoning tools for Semantic Web

Ontology reasoning tools have been built alongside the development of ontology languages. The rest of this subsection will introduce a few of these tools.

Cwm (Closed world machine) [18] is a general-purpose data processor for the Semantic Web. Implemented in Python and command-line based, it is a forward chaining reasoner for RDF.

Triple [17] is a RDF query, inference and transformation language. It does not have a built-in semantics for RDF Schema, but it allows semantics of languages to be defined with rules on top of RDF. This feature of Triple facilitates data aggregation as users can perform RDF reasoning and transformation under different semantics. The Triple tool supports OWL through external OWL reasoners such as FaCT and RACER.

FaCT (**Fa**st **C**lassification of **T**erminologies) [8], developed at University of Manchester, is a TBox (concept-level) reasoner that supports automated concept-level reasoning, namely class subsumption and consistency reasoning. It does not support ABox (instance-level) reasoning. It is implemented in Common Lisp and comes with a FaCT server, which can be accessed across network via its CORBA interface. Given an OWL ontology, it can classify the ontology (performs subsumption reasoning) so as to reduce redundancy and detect any inconsistency within it.

RACER, the **R**enamed **A**Box and **C**oncept **E**xpression **R**easoner [6], implements a TBox and partial ABox reasoner for the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})^-$ [7]. It can be regarded as (a) a Semantic Web inference engine, (b) a description logic reasoning system capable of both TBox and ABox reasoning and (c) a prover for modal logic

Km. In the Semantic Web domain, RACER's functionalities include developing ontologies (creating, maintaining and deleting concepts, roles and individuals); querying, retrieving and evaluating the knowledge base, etc. It supports OWL and RDF.

The FaCT and RACER are the most well accepted OWL reasoners. However, they still have many limitations, such as both of them can only flag an OWL class is inconsistent without providing any explanation. The debugging task is left to the user. Furthermore, there is very limited datatype support, such as integer and string. Also FaCT does not provide any ABox reasoning. RACER can only partially support ABox reasoning. Alloy approach proposed in this paper can complement FaCT and RACER [3].

Currently, there is not a well accepted system supporting SWRL, and only few prototypes has been developed. There is no reasoning tool supporting SWRL-FOL yet.

2.2 Alloy overview

Alloy [11] is a structural modelling language based on first-order logic, for expressing complex structural and behavioral constraints. Alloy treats relations as first class citizens and uses relational composition as a powerful operator to combine various structured entities. The essential constructs of Alloy are as follows:

- **Signature**: A signature (sig) paragraph introduces a basic type, a collection of relations (called field), and a set of constraints on their values. A signature may inherit fields and constraints from another signature.
- **Function**: A function (fun) captures behavior constraints. It is a parameterized formula that can be "applied" elsewhere.
- **Fact**: Fact ($fact$) imposes global constraints on the relations and objects. A $fact$ is a formula that takes no arguments and needs not to be invoked explicitly.
- **Assertion**: An assertion ($assert$) specifies an intended property. It is a formula whose correctness needs to be checked, assuming the facts in the model.

The Alloy Analyzer is a tool for analyzing models written in Alloy. Given a finite scope for a specification, Alloy Analyzer translates it into a propositional formula and uses SAT solving technology to generate instances that can satisfy the facts and properties expressed in the specification.

3 Alloy semantics for OWL/SWRL/SWRL-FOL

This section presents the Alloy semantics for OWL, SWRL and SWRL-FOL languages, which forms the foundation of the reasoning environment. Due to limited space, only part of semantic model has been presented here. A complete Alloy semantics for these languages can be found at `http://www.cs.man.ac.uk/~hwang/swrlfol.als`.

3.1 Alloy semantic for OWL constructs

3.1.1 Basic concepts

The semantic model for OWL is encoded in the module OWL. Users only need to import this module to reason about OWL ontology in Alloy.

```
module OWL
```

All the things described in the Semantic web context are referred to as web resources. A basic type `Resource` is defined as:

```
sig Resource {}
```

Other concepts such as classes and properties defined later are extended from the `Resource`. Property, which is a kind of Resource itself, relates Resource to Resource.

```
disj sig Property extends Resource
    {sub_val: Resource -> Resource}
```

"disj" is a keyward from Alloy for denoting the disjointness. Each Property has a relation sub_val from set <Property, Resource, Resource> with type <Resource, Resource, Resource> (since in Alloy subsignature does not introduce a new type). This relation can be regarded as a RDF statement, i.e., a triple of the form <property(or predicate), subject, value(or object)>.

The class corresponds to the generic concept of type or category of resource. Each Class maps a set of resources via the relation instances, which contains all the instance resources. The keyword disj is used to indicate the Class and Property are disjoint.

```
disj sig Class extends Resource {instances: set Resource}
```

The OWL also allows the use of XML Schema datatypes to describe (or define) part of the datatype domain. Alloy supports Integer and String. Apart from these there are no predefined types in Alloy, Datatype has been treated as a special Class, which contains all the possible datatype values in the instances relation.

```
disj sig Datatype extends Class {}
```

3.1.2 Class elements

The subClassOf is a relation between classes. The instances in a subclass are also in the super-classes. A parameterized formula (a function in Alloy) is used to represent this concept.

```
fun subClassOf(csup, csub: Class)
  {csub.instances in csup.instances}
```

The disjointWith is a relation between classes. It asserts that there are no instances common with each other.

```
fun disjointWith (c1, c2: Class) {no c1.instances & c2.instances}
```

3.1.3 Property restrictions

The allValuesFrom construct states that all instances of the class c1 that have the values of property P all belong to the class c2.

```
fun allValuesFrom
 (p: Property, c1: Class, c2: Class)
  {all r1, r2: Resource |
          r1 in c1.instances =>
          r2 in r1.(p.sub_val) =>
             r2 in c2.instances}
```

A hasValue function states that all instances of the class c1 have the values of property P as resource r. The r could be an individual object or a datatype value.

```
fun hasValue (p: Property, c1: Class, r: Resource)
    {all r1: Resource | r1 in c1.instances => r1.(p.sub_val) = r}
```

A MaxCardinality function states that all instances of the class c1 have at most N distinct values for the property P. Alloy supports some integer operations.

```
fun maxCardinality (p: Property, c1: Class, N: Int)
 {all r1: Resource| r1 in c1.instances <=>
     # r1.(p.sub_val) <= int N }
```

3.2 Boolean combination of class expressions

The intersectionOf function defines a relation between a class c1 and a list of classes clist. The List is defined in the Alloy library. The class c1 consists of exactly all the objects that are common to all class expressions from the list clist.

```
fun intersectionOf (clist: List, c1: Class)
     {all r: Resource| r in c1.instances <=>
         all ca: clist.*next.val | r in ca.instances}
```

The unionOf function defines a relation between a class c1 and a list of classes clist. The class c1 consists of exactly all the objects that belong to at least one of the class expressions from the list clist. It is analogous to logical disjunction;

```
fun unionOf (clist: List, c1: Class)
    {all r: Resource| r in c1.instances <=>
      some ca: clist.*next.val| r in ca.instances}
```

3.2.1 Property elements

The subPropertyOf construct states that psub is a sub-property of the property psup. This means that every pair (subject,value) that is in psup is also in the psub.

```
fun subPropertyOf (psup, psub: Property)
    {psub.sub_val in psup.sub_val}
```

The domain function asserts that the property P only applies to instances of the class c.

```
fun domain (p: Property, c: Class)
    {(p.sub_val).Resource in c.instances}
```

The inverseOf function shows two properties are inverse.

```
fun inverseOf (p1, p2: Property) {p1.sub_val = ~(p2.sub_val)}
```

All other OWL constructs can be defined in a similar manner. Please refer to the complete OWL Alloy semantics online.

3.3 Alloy semantic for SWRL extension

SWRL extends OWL by adding the rule axioms. A rule axiom consists of an antecedent and a consequent, each of which consists of a set of atoms. Atoms can be of the following forms, where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values.

- **C(x)**: Informally, it holds if x is an instance of the class description C.
- **P(x,y)**: It holds if x is related to y by property P.
- **sameAs(x,y)**: It holds if x is interpreted as the same object as y.
- **differentFrom(x,y)**: It holds if x and y are interpreted as different objects.

Table 2 shows how the above atoms can be modelled in Alloy.

Atom	**Alloy representation**
$C(x)$	'x in C.instances'
$P(x, y)$	'(x->y) in P.sub_val'
$sameAs(x, y)$	'x = y'
$differentFrom(x, y)$	'x != y'

Table 2: Alloy semantic for the atoms

As mentioned before, a rule means that if the antecedent holds, the consequent must also hold. It can be modelled as a universally quantified fact in the form of implication. For example the following rule axiom (where a_0 ... a_n are atoms)

$$Implies(Antecedent(a_1, ..., a_n)\ Consequent(a_0))$$

will be modelled as:

```
fact { a_1 && ... && a_n => a_0 }
```

SWRL-FOL formula	Alloy semantics
$and(C_1...C_n)$	fact $\{C_1$ && ... && C_n $\}$
$or(C_1...C_n)$	fact $\{C_1$ || ... || $C_n\}$
$neg(C)$	fact {not C}
$implies(C_1\ C_2)$	fact $\{C_1 => C_2\}$
$equivalent(C_1\ C_2)$	fact $\{C_1 <=> C_2\}$
$forall(V_1...V_n\ C)$	fact {all V_1, ..., V_n: Resource | C}
$exists(V_1...V_n\ C)$	fact {some V_1, ..., V_n: Resource | C}

Table 3: Alloy Semantic for SWRL-FOL

3.4 Alloy semantic for SWRL-FOL extension

SWRL-FOL extends SWRL with assertion axioms that contain first-order formulas. Table 3 presents the Alloy semantic for different SWRL-FOL formulas.

The above defines the basic transformation guidelines from the SWRL-FOL into their corresponding Alloy semantics. We will demonstrate the actual transformation process in the following section.

4 OWL/SWRL/SWRL-FOL to Alloy transformation

The previous section presented Alloy semantics for OWL, SWRL and SWRL-FOL, which forms the foundation for the reasoning environment. To be able to perform the automatic reasoning task using Alloy Analyzer, a Java program has been developed for the automatic transformation from an OWL/SWRL/SWRL-FOL knowledge file (in XML format) into its corresponding Alloy model.

A set of translation rules are developed in the following presentation.

4.1 OWL class translation

$$\frac{C \in OWL\_class}{static\ disj\ sig\ C\ extends\ Class\{\}}$$

An OWL_class C will be transferred into a scalar C, constrained to be an element of the signature Class.

4.2 OWL property translation

$$\frac{P \in OWL\_property}{static\ disj\ sig\ P\ extends\ Property\{\}}$$

An OWL_property p will be translated into a scalar P, constrained to be an element of the signature Property.

4.3 Instance translation

$$\frac{x \in instancesof[Y]}{\begin{array}{l} static\ disj\ sig\ x\ extends\ Resource\{\} \\ fact\{\ x\ in\ Y.instances\} \end{array}}$$

An OWL individual x of class Y will be translated into a scalar x, constrained to be an element of the signature Resource. x is a subset of Y.instances.

4.4 Other OWL translations

Other OWL constructs can be easily translated into the Alloy functions defined in the previous section. For example the following rule shows how to translate the OWL subclass relation into Alloy code.

$$\frac{subclass[X,\ Y], X \in OWL\_class,\ Y \in OWL\_class}{fact\{subClassOf(X,\ Y)\}}$$

4.5 SWRL and SWRL-FOL translation

The transformation of SWRL rules follows the semantics defined in Table 2. The variable x and y will be bound by some universal quantifiers. The SWRL rule can be modelled as a universally quantified fact in the form of implication.

Similarly, the transformation of SWRL-FOL follows the semantics presented in Table 3. More translation rules can be found from [19] and the web site `http://nt-appn.comp.nus.edu.sg/fm/alloy/introduction.htm`.

4.6 Translation example

The translation rules have been implemented in a Java program. The following OWL ontology defines two classes animal and plant which are disjoint. The eats and eaten_by are two properties, which are inverse to each other. The domain of eats is animal. The carnivore is a subclass of animal which can only eat animals. The ontology is given in a syntax similar to the "DL syntax" given in [10].

```
Class (animal)
Class (plant)
DisjointClasses( animal plant )
ObjectProperty(eaten_by)
ObjectProperty(eats
   domain (animal))
InverseProperties(eats eaten_by)
Class (carnivore complete animal
        restriction(eat allValuesFrom animal))
Class (herbivore complete animal
        restriction(eat allValuesFrom plant))
```

This fragment ontology can be transformed by the tool into the following Alloy segment.

```
module animal
/*import the defined library module */
open SWRL-FOL
/* plant and animal are translated to two class instances. The key
 word static is used to a signature containing exactly one element.*/
static disj sig plant, animal extends Class {}

/* The disjoin element was translated into fact in Alloy */
fact {disjointWith(plant, animal)}

/* eats, eaten_by are translated to two property instances */
static disj sig eats, eaten_by extends Property {}
fact {inverseOf(eats, eaten_by)}
fact {domain(eats, animal)}

static disj sig carnivore extends Class{}
fact{subClass(animal, carnivore)}
fact{allValuesFrom(eats, carnivore, animal)}
static disj sig herbivore extends Class{}
fact{subClass(animal, herbivore)}
fact{allValuesFrom(eats, herbivore, plant)}
```

The transformation of SWRL rules follows the semantics defined in Table 2. The variable x and y will be bound by some universal quantifiers. The SWRL rule can be modelled as a universally quantified fact in the form of implication. For example the following rule axiom

$$hasParent(?x1, ?x2) \wedge hasBrother(?x2, ?x3) \Rightarrow hasUncle(?x1, ?x3)$$

will be modelled as:

```
fact {all x1, x2, x3: Resource |
     (x1->x2) in hasParent.sub_val &&
     (x2->x3) in hasBrother.sub_val =>
          (x1->x3) in hasUncle.sub_val}
```

The transformation of SWRL-FOL follows the semantic presented in Table 3. After transforming the ontologties to the Alloy model, the consistency of the OWL/SWRL/SWRL-FOL ontology can be checked and some reasoning can be done readily.

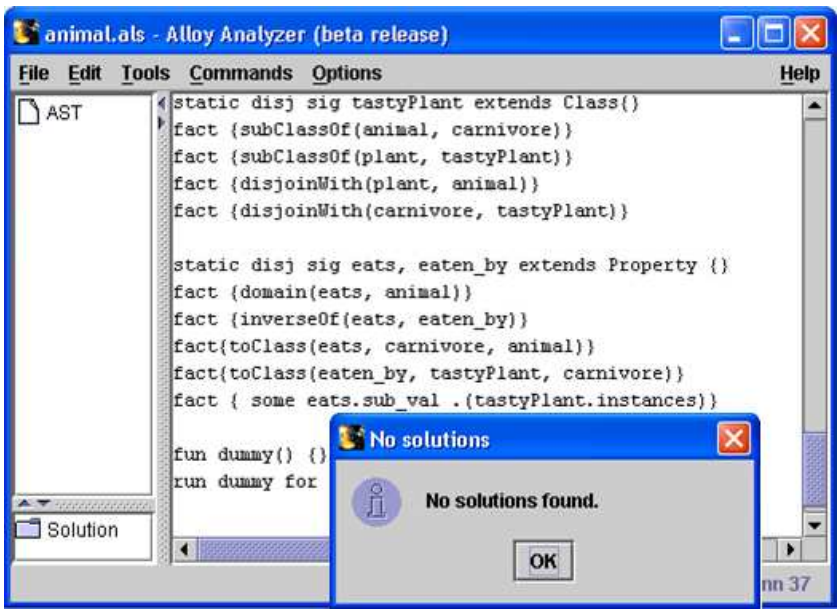

Figure 1: Inconsistence example

5 Reasoning OWL/SWRL/SWRL-FOL ontology with Alloy Analyzer

Reasoning is one of the key tasks for Semantic Web applications. It can be useful at many stages during the design, verification, maintenance and deployment of web ontology. In this section, we show that different Semantic Web reasoning tasks can be accomplished by using the Alloy Analyzer.

5.1 Standard OWL reasoning tasks

There are two different levels of checking and reasoning in OWL, the conceptual level and the instance level. At the conceptual level, the class properties and subclass relationships can be reasoned. At the instance level, the membership checking (instantiation) and instance property reasoning can be done.

5.1.1 Class property checking

It is essential that the ontology shared between autonomous software agents is conceptually consistent. Reasoning with inconsistent ontologies may lead to erroneous conclusions. An OWL class is deemed to be unsatisfiable (inconsistent) if, because of its description, it cannot possibly have any instances. This section gives some examples of inconsistent ontology that can arise in ontology development, and demonstrate how these inconsistencies can be detected by the Alloy Analyzer. For example, another

class tastyPlant which is a subclass of plant and eaten by the carnivore is defined. There is an inconsistency since by the ontology definition carnivores can only eat animals. Animals and plants are disjoint.

Class (tastyPlant partial plant
restriction(eat_by allValuesFrom(carnivore)))

We translate the ontology into an Alloy program, add some facts to remove the trivial models (like every type is empty set) and load the program into the Alloy Analyzer. The Alloy Analyzer will automatically check the consistency. AA attempts to find a model – a binding of the variables to values – that makes for the formulas (the formulas translated from the OWL model) true. If no such model can be build, it means that the model has been over constrained, i.e, there are some contradiction (inconsistency) in the model.

In the example, it can be concluded that there is an inconsistency in the animal ontology since Alloy can not find any solutions satisfying all facts within the scope (Figure 1). Note that when Alloy can not find a solution, it may be due to the scope being too small. By picking a large enough scope, "no solution found' is very likely to mean that an inconsistency has occurred. AA tried to constructs a model which satisfied all asserted axioms. If no such a model could be build ("no solution found"), then there are some contradicted axioms in the model.

Besides discovering the existence of an inconsistency in ontology, tracing where the inconsistency arises from is also crucial for a reasoning tool to be practical. The existing OWL reasoners like FaCT and RACER can only flag the inconsistent class without providing any explanation. The debugging process is left to users. Without any tool support, identifying the conflicting knowledge could be frustrating. One possible systematic technique for finding the causes of inconsistent ontology is to manually remove individual knowledge information until the culprit is identified. This task can be lengthy and dangerous.

In Alloy, the "unsatisfied core" [16] functionality of recent SAT solvers was utilized and it supports core extraction, a new analysis technique that helps to discover over-constraint in declarative models. This functionality can provide some assistance for the user to trace the inconsistency.

Extracting the unsatisfiable core of a CNF formula, that is a subset of the clause set sufficient to cause a contradiction, has been developed recently by satisfiability solvers [16]. In the latest version of Alloy, the declarative model analysis has been cast as satisfiability instances and the unsatisfiable core has been mapped back onto the model. In other words, a user can identify the parts of model responsible for producing the unsatisfiable CNF core. Those parts, by themselves, suffice to produce an over-constraint, and their identification can help the user find the over-constraint. Using this functionality, the portions of the ontology which contradict each other can be traced readily. In the animal example, suppose a new class named funnything was defined to be a subclass of both animal and plant classes. It is easy to see that there is an inconsistency since the class animal and plant are disjoint. Alloy can automatically identify a set of knowledge which makes the ontology unsatisfiable (Figure 2). The unsatisfiability maybe due to the fact that funnything is a subclass of animal, funnything is a

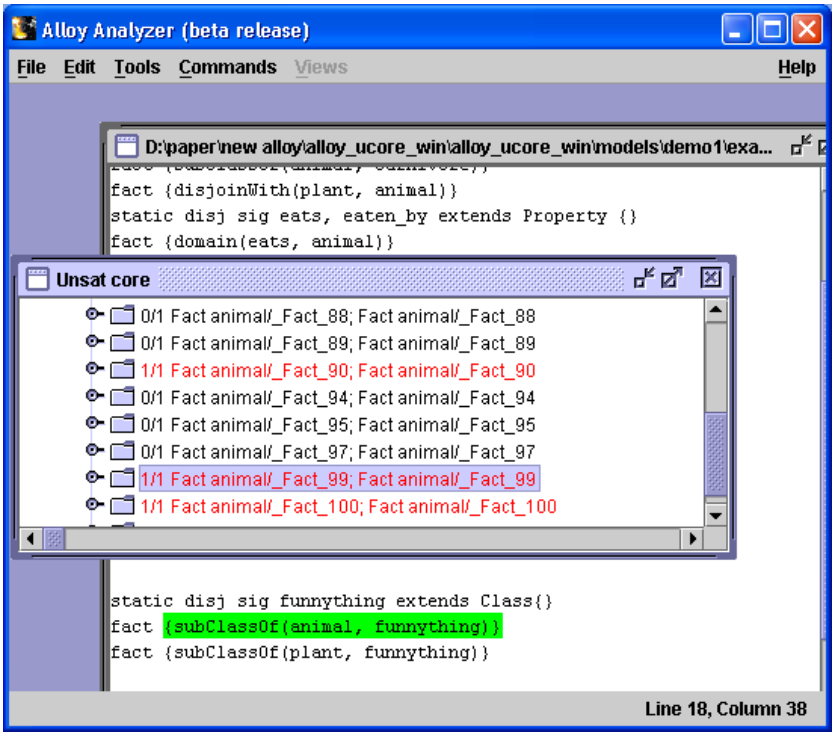

Figure 2: Tracing the inconsistency

subclass of plant or animal and plant are disjoint classes, and so on.

5.1.2 Subsumption reasoning

The task of subsumption reasoning is to infer an OWL class is the subclass of another OWL class. That is for every instances of one OWL class, it is an instance of another OWL class as well. Using AA, the subsumption relationship between classes can be checked automatically. The relationship between the fish, shark and dolphin has been used as an example to demonstrate this kind of reasoning task. In the animal ontology a property breathe_by is defined. The fish is a subclass of the animal which breathe_by the gill.

ObjectProperty(breathe_by)
Class (gill)
Class (fish complete animal
restriction(breathe_by allValuesFrom gill))

Since the purpose of this paper is to demonstrate ideas, the ontology has been kept simple. In reality there are some animals such as frogs and toads, which can respire by use of gills when they are young and by lungs when they reach adult stage. Also

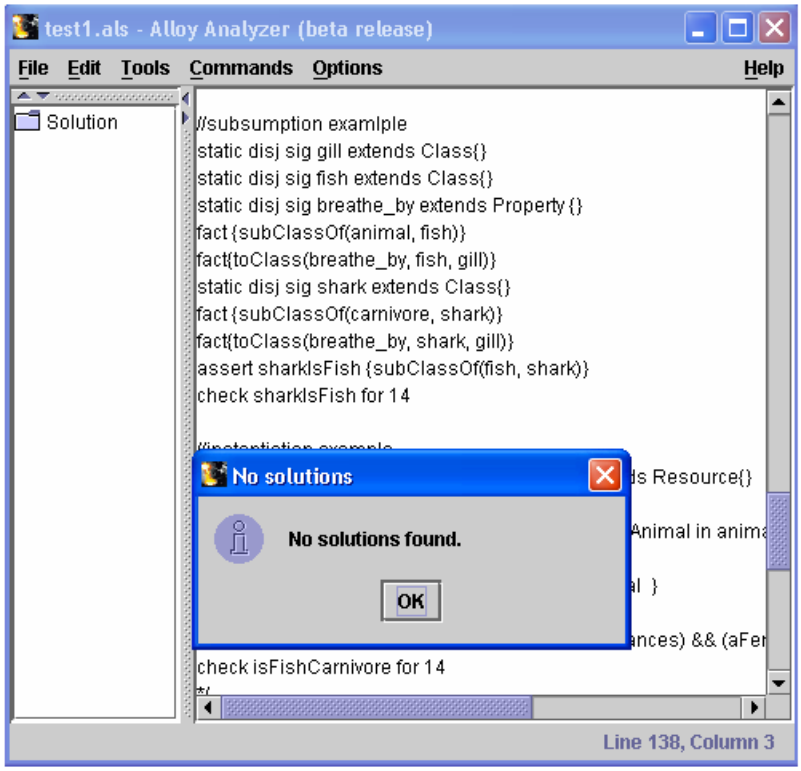

Figure 3: Subsumption example

cases like that the animals which respire by use of the pharyngeal lining or skin, like newborn Julia Creek dunnarts have not been considered. A class shark, a subclass of carnivore which breathe by the gill, has also been defined.

Class (shark)
Class (fish complete animal
restriction(breathe_by allValuesFrom gill))

Several of the classes were upgraded to be defined when their definitions constituted both necessary and sufficient conditions for class membership, e.g., an animal is a fish if and only if it breathes by the gill. Additional subclass relationships can be inferred, i.e., the shark is also a subclass of fish. We transfer this ontology into an Alloy program and make an assertion that the shark is a subclass of fish. The Alloy analyzer will check the correctness of this assertion automatically (Figure 3). The Alloy Analyzer checks whether an assertion holds by trying to find a counterexample. Note that "no solution" means no counterexample found, in this case, it strongly suggests that the assertion is sound. To make it more interesting, classes dolphin and lung are defined. Dolphins are a kind of animal which breathe by lungs. The classes gill and

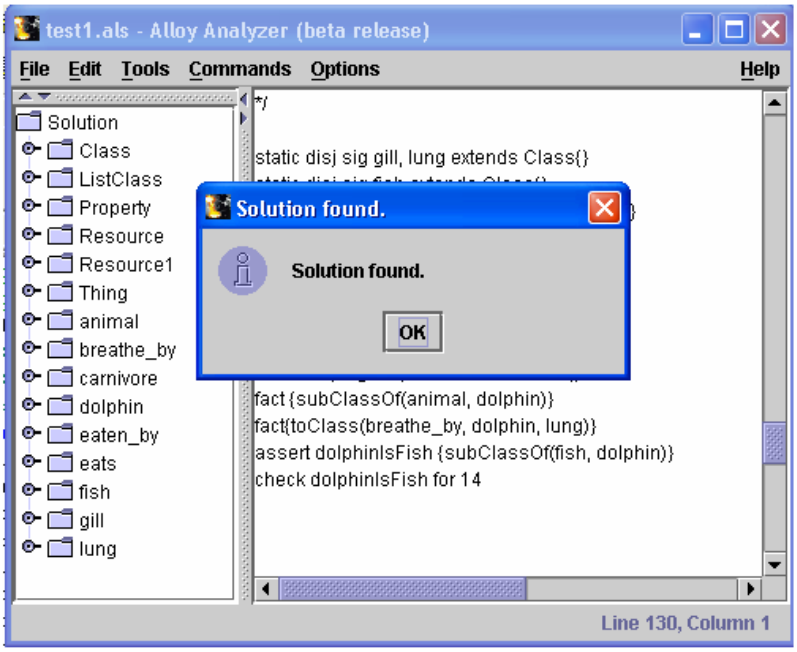

Figure 4: Dolphin is not a fish

lung are disjoint.

Class (lung)
DisjointClasses(lung gill)
Class (dolphin complete animal
restriction(breathe_by allValuesFrom lung))

Suppose an assertion that the dolphin is a kind of fish is made, the Alloy Analyzer will refute it since some counterexample was found (Figure 4). If the fact that dolphin is a fish is added in the module, the AA will conclude that an inconsistency has arisen.

5.1.3 Debugging uncompleted ontology

Information in OWL is gathered into ontologies, which can then be from different parties and stored as documents in the World Wide Web. Some knowledge may be missing in the ontology. Reasoning about uncompleted ontologies may lead to some unexpected results. We refer to the situation that because of some unavailable knowledges, the reasoners had inferred some unexpected knowledges which is different with natural facts. We need some tools to help the users to trace what is the missed knowledge causing the untrue conclusion that has been drew. AA checks the assertion by

generating counterexamples – structures or behaviors for which an expected property does not hold; from a counterexample, it is usually not too hard to figure out what is wrong. Looking at the counterexamples may provide some hints to the user on why the expected result does not hold and what knowledge is missing. For example, to show the OWL class dolphin and shark are disjoint, Intuitively, this is a correct statement since dolphin breathes by the gill while shark breathes by the lung. Gill and lung are disjoint. When the following assertion is added to Alloy, surprisingly AA concludes it is wrong.

```
assert disjointDS
   {disjointWith(shark, dolphin)}
```

By looking at the counterexamples graph, it has been noticed that all the counterexamples (an animal which is both a shark and a dolphin) generated by AA have empty values for the property breath by. In fact this unexpected result comes from the semantic of allValuesFrom construct in OWL. An OWL semantic can not deduce from a allValuesFrom restriction alone that there actually is at least one value for the property. An allValuesFrom restriction for a property is trivially satisfied for an instance that has no value for that property at all. The allValuesFrom restriction demands that all values of the property belong to a class, and if no such values exist, the restriction is trivially true. That is the reason why AA finds out the common instance, which does not breathe at all, for the class dolphin and class shark. To remove this expected result, extra knowledge needs to be added, e.g., an animal must breathe by something.

5.1.4 Instantiation

Instantiation is one of the main contributions for reasoning over OWL ontology using Alloy. Currently some successful OWL reasoners like FaCT are designed for description logic (DL) T-box reasoning, which lacks support for instances. In Alloy every expression denotes relations. The scalars will be represented by singleton unary relations - that is, relations with one column and one row. The instance level reasoning can be supported readily in Alloy.

Instantiation is a reasoning task which tries to check if an individual is an instance of a class. For example, two resources aFeralAnimal and aMeekAminal are defined as the instances of class animal. aGill is an instance of class gill. aFeralAnimal eats aMeekAnimal and breathes by aGill. People may want to check if aFeralAnimal is a carnivore and a fish.

Individual(`aMeekAnimal` type(animal))
Individual(`aGill` type(gill))
Individual(`aFeralAnimal` type(animal) value($breathe\_by$ `aGill`)
value($eats$ `aMeekAnimal`))

We translate the ontology into an Alloy program and make an assertion as following:

```
static disj sig aFeralAnimal, aMeekAnimal extends Resource{}
```

```
static disj sig aGill extends Resource{}
fact {aFeralAnimal in animal.instances &&
      aMeekAnimal in animal.instances}
fact {aGill in gill.instances}
fact {(aFeralAnimal->aMeekAnimal) in eats.sub_val}
fact {(aFeralAnimal->aGill) in breathe_by.sub_val}
assert isFishCarnivore
  {(aFeralAnimal in fish.instances)
   && (aFeralAnimal in carnivore.instances)}
check isFishCarnivore for 15
```

AA concludes that this assertion is correct.

5.1.5 Instance property reasoning

Instance property reasoning (often regarded as knowledge querying) is important in Semantic Web applications. It is a task to query some properties with individuals. Since one of the promising strengths of Semantic Web technology is that it gives the agents the capability to do more accurate and more meaningful searches. The agent can answer some questions for which the answers are not explicitly stored in the knowledge base.

For example, the emerge_early and emerge_later are two properties, which are inverse to each other. Animal A emerges earlier than B if the species of A emerge earlier than the species of B on the earth. emerge_early is transitive. Three animal instances firstDinosaur, firstApe and firstHuman are defined. firstDinosaur emerge_early than firstApe and firstApe emerge_early than firstHuman. One possible question people may ask is whether firstHuman is emerge_later than firstDinosaur. With the assistance of Alloy reasoner, such questions can be answered.

```
fact{TransitiveProperty(emerge_early)}
static disj sig firstDinosaur, firstApe,
  firstHuman extends Resource{}
fact { firstDinosaur in animal.instances
    && firstApe in animal.instances
    && firstHuman in animal.instances}
fact {(firstDinosaur->firstApe) in emerge_early.sub_val}
fact {(firstApe->firstHuman) in emerge_early.sub_val}
assert hum {(firstHuman->firstDinosaur) in emerge_later.sub_val}
check hum for 14
```

AA concludes that this assertion is correct.

5.2 SWRL/SWRL-FOL related reasoning

Besides of being capable to support the standard reasoning tasks on OWL, such as performing consistency checking, subsumption and instantiation reasoning automatically, moreover, Alloy can also check more complicated ontology properties expressed by the

newly extended languages such as SWRL/SWRL-FOL. In this section, we demonstrate how Alloy can be used to reasoning the SWRL-FOL ontologies.

A family relationship web ontology example is used here to illustrate the reasoning process. The following fragment of ontology first defines two OWL classes, `Person` and `twinParent` that represents the set of person who are the parents of twins, and three OWL object properties, i.e., `hasChild`, `brotherSister` and `sameBirthTime`. Secondly, the ontology class `wealthyParent` introduces the set of parents who have a child who is a `millionnaire`. Thirdly, two SWRL-FOL axiomatic assertions are defined to provide inference for the `brotherSister` and `twinParent` relationships. Lastly, the ontology class `wealthyTwinParent` is defined as a parent being both `wealthyParent` and `twinParent`.

```
Class (Person partial)
Class (twinParent partial Person)
Class (millionnaire partial Person)
ObjectProperty(hasChild)
ObjectProperty(brotherSister)
ObjectProperty(sameBirthTime)
Class (wealthyParent complete Person
       restriction(hasChild someValuesFrom(millionnaire)))
Assertion(forall I-variable(x1) I-variable(x2)
            (equivalent (exists (I-variable(x3)
            (and(hasChild(x3,x1) hasChild(x3,x2)
             differentFrom(x1,x2))))
            (brotherSister(x1, x2)))))
Assertion(forall I-variable(x1)
            (equivalent (exists (I-variable(x2)
            (exists (I-variable(x3)
            (and(brotherSister(x2, x3) sameBirthTime(x2, x3)
             hasChild(x1, x2))))))
            (twinParent(x1)))))
Class (wealthyTwinParent complete wealthyParent twinParent)
```

From the above, it is noticed that two SWRL-FOL axioms were asserted. The first assertion shows that if two distinct people have a same parent, then they are brothers or sisters. The second assertion in the above ontology shows that if two people are brothers or sisters, and they have the same birth time, then their parents are twin-parents. Furthermore, suppose some instances of the above ontology are asserted into the knowledge base as follows.

```
Individual(Tom type(person)
     type(complementOf(wealthyTwinParent))
     value(hasChild Jerry)
     value(hasChild Jim))
Individual(Jerry type(millionnaire) value(sameBirthTime Jim))
     Individual(Jim type(person))
     DisjointWith(Jim Jerry)
```

We transform the above ontology (in XML format) into its Alloy model[3] using our transformation program.

[3]Due to the space limit, the complete Alloy model of the above family relationship ontology example can be found at `http://www.cs.man.ac.uk/~hwang/FAMILY.als`.

Similar as reasoning OWL, the Alloy Analyzer can automatically perform different reasoning tasks for SWRL/SWRL-FOL. For example, it can detect that there is an inconsistency in the above ontology example, as the Alloy Analyzer can not find any ontology instances (solutions) satisfying all facts within the scope.

In this family ontology example, the inconsistency comes from the fact that `Tom` has been inferred as an instance of both the class `wealthyParent` and the class `twinParent`. However, there is a piece of knowledge in the model that explicitly indicates that `Tom` is not an instance of the `wealthyTwinParent` class, which contradicts to the inferred conclusion. As discussed before, with the assistance of Alloy Analyzer's "unsatisfiable core" functionality, the debugging process of identifying the source of inconsistency in the ontology becomes much more handy to the users.

5.3 Discussion

The correctness of the translation has been verified by many different test cases. A same problem has been sent to existing SW tools, theorem provers and Alloy; the same conclusions are drawn. Furthermore, the OWL has well defined semantics in first order logic and Alloy is also based on the first-order logic. The soundness of the translation can also be proved easily. In the early work [13], it shows that the consistence between the Alloy Semantic for the Semantic Web languages and the original OWL semantic (Alloy has been regarded as a subset of Z). Formal proving this consistence is beyond the scope of this paper.

6 Related works and conclusion

This paper presented a reasoning environment for the Semantic Web ontology family languages (OWL/SWRL/SWRL-FOL). There are four main contributions of the paper. Firstly, it defines a semantic encoding for the OWL/SWRL/SWRL-FOL constructs in the Alloy first-order language. Secondly, it presents a systematic transformation tool from the OWL/SWRL/SWRL-FOL ontology (in XML) into its corresponding Alloy model. Thirdly, with the assistance of Alloy Analyzer, it has been demonstrated that the consistency of an ontology model can be checked automatically and different kinds of reasoning tasks can be supported. Our approach complements with existing OWL reasoners by providing full automatic debugging aids and instance level reasoning. Furthermore, SWRL-FOL is a newly proposed extension to OWL, and to our best of knowledge, so far there is no existing reasoning support for SWRL-FOL prior to this work. Finally, the paper also demonstrates a light-weight formal methods approach to the web ontology domain. Alloy was chosen over other reasoning tools because it is based on first-order relational logic and relations between Web resources are the focus issues in the Semantic Web context. Furthermore, Alloy has an impressive automatic tool support, the Alloy Analyzer, where automated generation of finite set of ontology instances, creation of counter-examples on assertions, and identifying the source of inconsistencies in the model are made available. Usual ontology tools such as FACT and RASER can detect errors in an ontology model, but may not be able to point out where the error is. Alloy approach provides the ontology "surgery" like capability to

pin point the errors in the model with counter-examples or contradictory constraints. This is a highly complementary approach to Semantic Web reasoning. The approach has been successfully applied to a recent military ontology [5].

It has indeed been realized that there is a limitation on the scalability of the current Alloy Analyzer in reasoning large ontology models. The approach presented here can only deal with the ontologies with relatively small size. Based on the same idea, authors also attempt to use the theorem prover, i.e. Z/EVES, to reason the SW ontology [4]. The theorem prover can handle large sized ontologies, but it requires the user's interaction. Here authors do not claim that Alloy is the only and best formal tool to reason over SW ontologies, but authors do claim that it is an effective attempt with certain novel and irreplaceable advantages like full automation and promising debugging assistance. In fact, it is unlikely in the near future that both expressive and automatic tool will be developed. Currently, it is desirable if the strength from different ontology reasoning tools can be integrated. [3] presented the methodology of checking ontologies using tools RACER, Z/EVES and AA in conjunction. This approach has been successfully applied for reasoning a real life military ontology.

In the future, it has been planned to integrate the current Alloy Analyzer reasoning facilities into our OWL/SWRL/SWRL-FOL transformation tool by connecting it to the Alloy API interfaces. In addition, we also plan to extend the transformation tool with the editing and designing functions for the ontology models, so that it will become an integrated development environment for the web ontology modelling, which includes design, transformation and reasoning functions in one coherent tool support.

Acknowledgements

This work was supported in part by the HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council.

References

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

[2] D. Brickley and R.V. Guha (editors). Resource description framework (rdf) schema specification 1.0. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, March, 2000.

[3] J. S. Dong, C. H .Lee, Y. F. Li, and H. Wang. A combined approach to checking web ontologies. In *The 13th ACM International World Wide Web Conference (WWW'04)*, pages 714–722. ACM Press, May 2004.

[4] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and Beyond in Z/EVES. In *Proc. The 26th International Conference on Software Engineering (ICSE'04)*, pages 201–210, Edinburgh, Scotland, May 2004.

[5] J. S. Dong, J. Sun, H. Wang, C. H. Lee, and H. B. Lee. Analysing Web Ontology in Alloy: A Military Case Study. In *Proc. 15th International Conference on*

Software Engineering and Knowledge Engineering: SEKE'2003, pages 542–546, San Francisco, USA, July 2003.

[6] Volker Haarslev and Ralf Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701–705, 2001.

[7] Volker Haarslev and Ralf Möller. Practical Reasoning in Racer with a Concrete Domain for Linear Inequations. In Ian Horrocks and Sergio Tessaris, editors, *Proceedings of the International Workshop on Description Logics (DL-2002)*, Toulouse, France, April 2002. CEUR-WS.

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

[9] I. Horrocks, P.F. Patel-Schneider, H. Boley, S.Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Available: http://www.daml.org/2003/11/swrl/, 2003.

[10] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[11] D. Jackson. Micromodels of software: Lightweight modelling and analysis with alloy. Available: http://sdg.lcs.mit.edu/alloy/book.pdf, 2002.

[12] O. Lassila and R. R. Swick (editors). Resource description framework (rdf) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Feb, 1999.

[13] Dorel Lucanu, Yuan Fang Li, and Jin Song Dong. Institution Morphisms for Relating OWL and Z. In *The 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, Taipei, Taiwan, July 2005.

[14] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Available: http://www.w3c.org/TR/owl-features/, 2004.

[15] P.F. Patel-Schneider, P. Hayes, I. Horrocks, and F. Harmelen. A Proposal for a SWRL Extension to First-Order Logic. Available: http://www.daml.org/2003/11/swrl/, 2004.

[16] Ilya Shlyakhter, Robert Seater, Daniel Jackson, Manu Sridharan, and Mana Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. In *Proc. 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pages 94–105, Montreal, Quebec, Canada, October 2004.

[17] Michael Sintek and Stefan Decker. TRIPLE—A query, inference, and transformation language for the semantic web. In I. Horrocks and J. Hendler, editors, *The Semantic Web — ISWC 2002. Proceedings of the First International Semantic Web Conference*, volume 2348 of *Lect. Notes in Comput. Sci.*, pages 364–378, Sardinia, Italy, June 2002. Springer-Verlag.

[18] Tim Berners-Lee. cwm - a general purpose data processor for the semantic web. `http://www.w3.org/2000/10/swap/doc/cwm`, 2004.

[19] Hai Wang, Jin Song Dong, and Jing Sun. Reasoning Support for SWRL-FOL Using Alloy. In *17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, Taipei, Taiwan, July 2005.

Biographical Notes

Hai H. Wang obtained Bachelor (1st class honors) and PhD degrees from the School of Computing, National University of Singapore (NUS) in 2001 and 2004. He worked as a Research Assistant in the School of Computing at NUS from 2001-2003. Since 2004 he has been in the School of Computer Science at The University of Manchester where he worked as a Research Associate. His main interests include Software Engineering, Formal Methods Ontology and Semantic Web.

Jin-Song DONG received Bachelor (1st class honors) and PhD degrees in Computing from University of Queensland in 1992 and 1996. From 1995-1998, he was a Research Scientist at the Commonwealth Scientific and Industrial Research Organisation in Australia. Since 1998 he has been in the School of Computing at the National University of Singapore (NUS) where he is currently Associate Professor and Assistant Dean. He is a Steering Committee member of the International Conference on Formal Engineering Methods (ICFEM) and the Asia Pacific Software Engineering Conference (APSEC) series.

Jing Sun is a lecturer at the Department of Computer Science, The University of Auckland, New Zealand. He obtained his PhD degree from the Department of Computer Science, National University of Singapore in March 2004. Dr. Sun's research interests include Software Engineering, Formal Methods and Semantic Web.

Jun Sun received the BSc degree from the School of Computing, National University of Singapore (NUS) in 2002. Since then he has been pursuing the PhD degree in software engineering from NUS. As of July 206, he is a research fellow in the department of Computer Science at NUS.