Verifying OWL and ORL Ontologies in PVS

Jin Song Dong, Yuzhang Feng*, and Yuan Fang Li

School of Computing, National University of Singapore
{dongjs, fengyz, liyf}@comp.nus.edu.sg

Abstract. The Semantic Web vision is being realized to reach the full potential of the Web. Semantic data modeling is the foundation of the Semantic Web. The Web Ontology Language (OWL) and OWL Rules Language (ORL) provides basic machinery to the semantic mark-up for data. However, there is limited tool support for OWL and no tool support currently for ORL. In this paper, we propose to model OWL and ORL language semantics in PVS specification language so that OWL and ORL ontologies can be transformed and verified in the Prototype Verification System (PVS). PVS user-defined proof strategies are also developed to automate the proof process.

Keywords: PVS, Semantic Web, OWL, ORL, reasoning.

1 Introduction

Unlike conventional web as we have now, the Semantic Web (SW) [2] is a platform for inter-machine data and information exchange, filtering, integration, etc., across organizational boundaries without human supervision. It extends the current web and reaches its full potential by making it truly ubiquitous and ready for the machines. The Web Ontology Language (OWL) [7], a Recommendation by World Wide Web Consortium (W3C), defines the basic vocabulary for describing data on the web and is a layer on which Web Services can be developed. In a way, modeling of data using OWL is an important part of requirements engineering for Semantic Web.

In order for intelligent software agents to automatedly process data on the web, ontology languages such as DAML+OIL and (part of) OWL were originally designed to be decidable [19, 22]. However, the trade-off is the limited expressiveness, which forbids some very desirable properties to be specified. To partially overcome this limitation of OWL, the OWL Rules Language (ORL) [12] has recently been proposed by Horrocks & Patel-Schneider.

Reasoning tool support for OWL is limited at the moment. Moreover, currently there is no tool support for ORL. SW reasoning tools such as FaCT [11] and RACER [10] have been developed to be fully automated; hence they cannot support ORL without major modification. However, as it can be foreseen

* Author for correspondence: fengyz@comp.nus.edu.sg.

Z. Liu and K. Araki (Eds.): ICTAC 2004, LNCS 3407, pp. 265–279, 2005.

that ORL be integrated into the ontology languages hierarchy, the correctness of OWL and ORL ontologies is crucial to establishing *trust* in Semantic Web.

SW can be regarded as an emerging area from the knowledge representation and the web communities. The software engineering community can also play an important role in the SW development. Software verification techniques can be applied to check SW ontology related properties. We believe SW will be a new research and application domain for software engineering, especially for software verification techniques. In this paper, we propose to develop reasoning environment in PVS for OWL and ORL.

The rest of the paper is organized as follows. We briefly introduce the Semantic Web, ontology languages, tools and PVS in Section 2. In Section 3, we present PVS semantics for OWL with ORL axioms. In Section 4, we concisely discuss transformation from ORL to PVS. Reasoning support for OWL and ORL using PVS theorem prover is presented through a few case studies in Section 5. Section 6 presents related works, summarizes our contribution and discusses possible future works.

2 Overview

2.1 Semantic Web, Languages and Tools

Semantic Web. Although the traditional World Wide Web (WWW) was originally designed for machine processing, it ends up to be consumed only by human, i.e., web contents are only visually marked-up for humans to read. To reach the its full potential, it is necessary to make the web a platform for intelligent software agents to interact with each other to accomplish complex tasks without human supervision. To achieve this goal, data on the web must be given structured and precise meaning so that software agents can process data cooperatively and autonomously. The Semantic Web [2] was proposed by Tim Berners-Lee as the next generation of the web and it is now a W3C activity in its second phase.

Ontology Languages. Data in SW are represented by ontologies, which define their concepts and relationships. Ontology languages provide vocabularies for expressing ontologies.

Built on top of XML, the Resource Description Framework (RDF) [14] is a model of metadata defining a mechanism for describing resources without assumptions about a particular application domain. RDF describes web resources in a simple triplet format: $\langle subject\ predicate\ object \rangle$, where *subject* is the resource of interest, *predicate* is one the properties of this resource and *object* states the value of this property. RDF Schema [4] provides facilities to describe RDF data. RDF Schema allows structured and semi-structured data to be mixed together, which makes them hard for machines to process.

The syntactic ambiguity and relatively limited expressiveness of RDF Schema is partially overcome by the DARPA Agent Markup Language (DAML) [19], which is built on top of RDF Schema and based on description logics. DAML pooled effort with the Ontology Inference Layer project [5] to produce the ontol-

ogy language DAML+OIL. It provides a richer set of language primitives to describe classes and properties than RDF Schema and allows only structured data.

In 2004, a new ontology language based on DAML+OIL, the Web Ontology Language (OWL) [21] became the W3C Recommendation. It consists of three sublanguages: OWL Lite, DL & Full, with increasing expressiveness. These languages are designed for user groups with different requirements. OWL Lite & DL are decidable but Full is generally not. The undecidability of OWL Full comes from relaxing certain constraints from OWL DL. For example, OWL Full does not enforce the mutual exclusiveness between classes, properties, data values and individuals.

Although the design of OWL has taken into consideration of the different expressiveness needs of different user groups, it is still not expressive enough. Some very desirable properties cannot be expressed even in OWL Full. An important reason for this is that although the language provides a relatively rich set of language primitives for describing classes, it does not provide as many primitives for describing properties. For example, it does not support property composition. In the light of this weakness, Horrocks and Patel-Schneider [12] proposed an extension to OWL, the OWL Rules Language (ORL), in a syntactically and semantically coherent manner. ORL incorporates Horn clause rules into OWL and makes rules part of axioms that can be used to express more complex classes and properties.

The major extensions of ORL are the inclusion of Horn clause rules and variable declarations. The rules are in the form of $\texttt{antecedent} \rightarrow \texttt{consequent}$, where both antecedent and consequent are conjunctions of atoms: class membership, property membership, individual (in)equalities. Informally, a rule means that if the antecedent holds, then the consequent must also hold. A simple example rule shown below states that if $?b$ is a parent of $?a$ and $?c$ is a brother of $?b$, then $?c$ is an uncle of $?a$.

$$\texttt{parent}(?a, ?b) \wedge \texttt{brother}(?b, ?c) \rightarrow \texttt{uncle}(?a, ?c)$$

Ontology Tools. Various ontology tools have been built to support the development of the SW, such as ontology design, creation, management, merging, maintenance, publishing, reasoning, etc. In the rest of this section, we will briefly introduce a few reasoning tools.

FaCT (**Fa**st **C**lassification of **T**erminologies) [11] is a description logics classifier developed at University of Manchester. FaCT supports automated concept-level reasoning (concept subsumption and satisfiability testing), but not instance-level reasoning. Currently FaCT supports DAML+OIL and OWL.

RACER (**R**enamed **A**Box and **C**oncept **E**xpression **R**easoner) [10] is a reasoner for the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})^-$ [9]. It has a much richer set of functionalities compared to that of FaCT, including ontology creation, query, retrieval and evaluation, knowledge base conversion to DAML+OIL/OWL, etc.

2.2 PVS

The Prototype Verification System (PVS) is an integrated environment for the development of formal specifications written in the PVS specification language

[17]. It supports a wide range of activities in specification development: creation, documentation, type-checking, theorem-proving, etc. The distinguishing feature of PVS is its synergistic integration of an very expressive specification language and powerful theorem-proving capabilities.

PVS provides an expressive specification language that augments classical higher-order logic with a sophisticated type system with predicate subtypes and dependent types, and with parameterized theories and a mechanism for defining abstract data types such as lists and trees.

PVS specifications are organized into *theories*, which define data types, axioms, theorems and conjectures that can be reused by other theories.

PVS has a powerful interactive theorem prover/proof checker [16]. The basic deductive steps in PVS are large compared with many other systems: there are atomic commands for induction, quantifier reasoning, automatic condition rewriting, simplification, etc. User-defined proof strategies can be used to enhance the automation in the proof checker.

The proof goal in PVS is represented as a sequent which consists of a list of formulas called antecedents and a list of formulas called consequents. The interpretation of a sequent is that the conjunction of the antecedents implies the disjunction of the consequents. Either or both of the antecedents and consequents may be empty. An empty antecedent is equivalent to true, and an empty consequent is equivalent to false, so if both are empty the sequent is false. Every proof in PVS starts with a single consequent. It can be seen that the structure of sequents in PVS very much resembles that of the rules in ORL except that in ORL the conjunction of antecedents implies the conjunction of consequents. But as pointed out in [12] that an ORL rule of multiple consequents can be easily transformed into multiple rules each with a single consequent. Therefore we believe PVS is a natural reasoner for ORL.

3 PVS Semantics for OWL and ORL

In order to use PVS to verify and reason ontologies with ORL axioms, it is necessary to define the PVS semantics for OWL & ORL. This semantic model forms the reasoning environment for verification using PVS theorem prover. In this section, we present a PVS specification for a subset of OWL Full language primitives and the newly proposed ORL. The complete model can be found online[1].

3.1 PVS Semantics for OWL Constructs

Basic Concepts

Everything in Semantic Web is a *Resource*. So we model it by defining a non-empty type in PVS.

```
RESOURCE: TYPE+
```

[1] `http://www-appn.comp.nus.edu.sg/~rpfm/ORL2PVS`

In OWL Full the universe of individuals consists of all resources. Thus we define *Individual* to be a type equivalent to *Resource*.

```
INDIVIDUAL: TYPE+ = RESOURCE
```

Each class in OWL is a resource, which has a number of individuals associated with it: the instances of this class. So we model *Class* as a subtype of *Resource* and define a function *instances* that maps a class to a set of individuals.

```
CLASS: TYPE+ FROM RESOURCE
instances: [CLASS -> set[INDIVIDUAL]]
```

A property relates resources to resources. So we model *Property* as a predicate over a tuple of two resources.

```
PROPERTY: TYPE = pred[[RESOURCE,RESOURCE]]
```

Class Relationships

The property *subClassOf* is defined as a boolean function from two classes. For a class `c1` to be the sub-class of class `c2`, the instances of `c1` must be a subset of the instances of `c2`.

```
subClassOf?(c1,c2:CLASS): bool =
(
    subset?(instances(c1),instances(c2))
)
```

Other class relationship properties such as *disjointWith* and *equivalentClass* are similarly defined.

Class and Property

The property *allValuesFrom* attempts to establish a maximal set of individuals as a class. It defines a class `c1` of all individuals `i1` for which it holds that if the pair `(i1,i2)` is in the property `p` implies that `i2` is an instance of class `c2`. So we model it as a function from a property `p` and a class `c2` to a class `c1` and specify its meaning as an axiom as follows.

```
allValuesFrom: [PROPERTY, CLASS -> CLASS]
allValuesFrom_ax: AXIOM FORALL (c1,c2:CLASS),(p:PROPERTY):
    (allValuesFrom(p,c2) = c1 IMPLIES FORALL (i1:INDIVIDUAL):
        member(i1,instances(c1)) IFF FORALL (i2:INDIVIDUAL):
            (p(i1,i2) IMPLIES member(i2,instances(c2)))))
```

Property Relationships

The property *subPropertyOf* states that a property $p1$ is a sub-property of property $p2$ if and only if all pairs $(i1, i2)$ in $p1$ are also in $p2$. Therefore it is modeled as a boolean function of two properties.

```
subPropertyOf?(p1,p2:PROPERTY): bool =
(
    FORALL (i1,i2:INDIVIDUAL):  (p1(i1,i2) IMPLIES p2(i1,i2))
)
```

3.2 ORL Extension

In ORL [12], a rule consists of an antecedent and a consequent, each of which consists of a (possibly empty) set of atoms. Atoms can be of the form $C(x)$, $P(x, y)$, $sameAs(x, y)$ or $differentFrom(x, y)$, where C is an OWL class description, P is an OWL property, and x, y are either variables, OWL individuals or OWL data values. Informally, an atom $C(x)$ holds if x is an instance of the class description C, an atom $P(x, y)$ holds if x is related to y by property P, an atom $sameAs(x, y)$ holds if x is interpreted as the same object as y, and an atom $differentFrom(x, y)$ holds if x and y are interpreted as different objects. A rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold.

An ORL rule will be modeled as a PVS rewrite rule, e.g., a universally quantified predicate of the form

$$a_1 \wedge a_2 \wedge ... \wedge a_m \Rightarrow c_1 \wedge c_2 \wedge ... \wedge c_n$$

where a_i and c_j are one of the four forms of atoms.

3.3 Proof Support for PVS

To make the proving process of PVS more automated, a set of rewrite rules and theorems is also defined. They aim to hide certain amount of underlying model from the verification and reasoning and to achieve abstraction and automation. Usually these rules relate several classes & properties by defining the effect of using them in a particular way. One simple example is the `subClassOf_transitive` theorem. It states that if a class `c1` is a sub-class of a class `c2` and `c2` is a sub-class of a class `c3`, then `c1` is a sub-class of `c3`.

```
subClassOf_transitive: THEOREM FORALL (c1,c2,c3:CLASS):
    subClassOf?(c1,c2) AND subClassOf?(c2,c3) IMPLIES
       subClassOf?(c1,c3)
```

The following theorem, *member_subClassOf* states that an instance of a particular class is also an instance of all the super classes of this class.

```
member_subClassOf: THEOREM
    FORALL (i:INDIVIDUAL),(c1,c2:CLASS):
        member(i, instances(c1)) AND subClassOf?(c1,c2)
            IMPLIES member(i, instances(c2))
```

4 Transforming ORL to PVS

As ORL is an extension to OWL with the inclusion of rules, we perform the transformation in two steps. We transform OWL constructs into PVS specifications first, followed by the transformation of ORL rules.

We have developed a tool in Java to automatically transform OWL ontologies into PVS specifications. For example, the following ontology fragment defines a class *Person* and specifies some of its properties. The transformed PVS fragment is shown at the right.

```
<owl:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf><owl:Restriction>
      <owl:onProperty
         rdf:resource="#hasParent"/>
      <owl:allValuesFrom
         rdf:resource="#Person"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction owl :cardinality="1">
      <owl:onProperty
         rdf:resource="#hasFather"/>
    </owl:Restriction></rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class prefab="#Man"/>
    <owl:Class prefab="#Woman"/>
  </owl:unionOf></owl:Class>
```

```
Person: CLASS
Person_union_ax:
    AXIOM Person=unionOf((:Man,Woman:))
Person_subClassOf_ax_1:
    AXIOM subClassOf?(Person,Animal)
Person_subClassOf_ax_2: AXIOM subClassOf?
    (Person,allValuesFrom(hasParent,Person))
Person_subClassOf_ax_3: AXIOM subClassOf?
    (Person,cardinality(hasFather,1))
```

In order to facilitate reasoning about numbers, data type properties are transformed into predicates and functional data type properties are transformed into functions. The advantage of doing this will become clearer when we discuss reasoning in Section 5. For example, transformation of the datatype property *age* is given below the OWL fragment:

```
<owl:DatatypeProperty rdf:ID="age">
  <rdf:type rdf:resource="http://www.w3.org/
    2002/07/owl#FunctionalProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/
    2000/10/XMLSchema#nonNegativeInteger"/>
</owl:DatatypeProperty>

age: [INDIVIDUAL -> Nat]
```

The tool we developed is also capable of transforming instance ontologies into PVS specifications. For example, the following shows an OWL instance ontology fragment and the corresponding PVS specification.

```
<Description rdf:ID="Ian">
  <rdf:type>
    <owl:Class rdf:ID="Person"/>
  </rdf:type>
  <shoe size>14</shoe size>
  <age>37</age>
</Description>
```

```
Ian: INDIVIDUAL
Ian_Person_ax:
    AXIOM member(Ian,instanceOf(Person))
Ian_shoesize_14_ax: AXIOM shoe_size(Ian)=14
Ian_age_37_ax: AXIOM age(Ian)=37
```

Transformation of ORL rules is straightforward. Each rule is transformed into an axiom, which is a universally quantified Horn clause with each of the atoms transformed into a predicate. For example,

```
<owlr:Rule rdf:ID="Rule1">
  <owlr:antecedent>
    <owlr:individualPropertyAtom  polypro="hasParent">
      <owlr:Variable tournament="x1" />
      <owlr:Variable tournament="x2" />
    </owlr:individualPropertyAtom>
    <owlr:individualPropertyAtom  polypro="hasBrother">
      <owlr:Variable tournament="x2" />
      <owlr:Variable tournament="x3" />
    </owlr:individualPropertyAtom>
  </owlr:antecedent>
  <owlr:consequent>
```

```
    <owlr:individualPropertyAtom  polypro="hasUncle">
      <owlr:Variable tournament="x1" />
      <owlr:Variable tournament="x3" />
    </owlr:individualPropertyAtom>
  </owlr:consequent>
</owlr:Rule>
```

is transformed into

```
Rule1_ax: AXIOM FORALL (x1,x2,x3: RESOURCE)
    hasParent(x1,x2) AND hasBrother(x2,x3)
        IMPLIES hasUncle(x1,x3)
```

5 Ontology Reasoning Using PVS

In this section, we demonstrate how PVS can be used to check ontology-related properties and to reason beyond the modeling power of OWL & ORL. It is presented in two parts. Firstly, standard SW reasoning are performed. In the second part, we show how PVS can reason ORL and more complex properties that even ORL cannot express.

5.1 Standard SW Reasoning

Standard SW reasoning includes three categories, namely inconsistency checking, subsumption reasoning and instantiation reasoning. The following subsections illustrate each category with an example.

Inconsistency Checking. Ensuring the consistency of ontologies is an important task in various stages of ontology development, as inconsistent ontologies may lead agents to reason erroneously and make wrong conclusions.

To be precise, knowledge base consistency amounts to verifying whether every concept in the knowledge base admits at least one individual [15].

The following is an example of inconsistency checking in the animal ontology. After transforming the ontology into a PVS specification, we identified the following closely related classes, properties and their axioms.

```
Animal,Vegetarian,Cow,MadCow,Food,Meat,Vegetable:CLASS
eats:O_PROPERTY
Vegetarian_subClassOf_ax_1: AXIOM subClassOf?(Vegetarian,Animal)
Vegetarian_allValuesFrom_ax_1: AXIOM Vegetarian=allValuesFrom(eats,Vegetable)
Cow_subClassOf_ax_1: AXIOM subClassOf?(Cow,Vegetarian)
MadCow_subClassOf_ax_1: AXIOM subClassOf?(MadCow,Cow)
MadCow_subClassOf_ax_2: AXIOM subClassOf?(MadCow,someValuesFrom(eats,Meat))
Meat_subClassOf_ax_1: AXIOM subClassOf?(Meat,Food)
Vegetable_subClassOf_ax_1: AXIOM subClassOf?(Vegetable,Food)
Vegetable_disjointWith_ax_1: AXIOM disjointWith?(Vegetable,Meat)
```

We suspect that there is an inconsistency in the class of *MadCow*. To prove that, we assert the following theorem, which means that the class of *MadCow* does not admit any individual.

```
MadCow_inconsistent: THEOREM
    (EXISTS (i:INDIVIDUAL):member(i, instances(MadCow))) IMPLIES FALSE
```

After applying `(lemma)` to supply PVS with known facts (axioms), applying `(skolem!)` to remove quantifiers and instructing PVS to understand the subclass relationship between *MadCow* and *Vegetarian*, we need to prove `member(i!1,instances(Vegetarian))`, that $i!1$ is a member of *Vegetarian*, which can be proved by the theorem *member_subClassOf* introduced in Section 3.3.

By expanding the definition of *Vegetarian* and exploiting the fact that *MadCow* is a subclass of an anonymous class that *eats Meat*, we can finish up the proof using a `(grind)`, which is a catch-all strategy that is frequently used to automatically complete a proof branch or to apply all the obvious simplifications.

Subsumption Reasoning. The task of subsumption reasoning is to infer that an OWL class is a sub-class of another. This could be accomplished in PVS fairly automatically. One of the simplest ways is by using the fact that `subClassOf?` is a transitive property, which can be easily proved by PVS.

There are other ways of proving subsumption relationships. One of them is by inter-class relationships such as `intersectionOf` and `UnionOf`. For example, we have the following transformed ontology fragment and we want to prove that the class *TallMan* is a subclass of *Person* using theorem *TallMan_subClassOf_Person* defined on the right:

```
TallMan_intersection_ax: AXIOM
  TallMan=intersectionOf((:TallThing,Man:))
Person_union_ax: AXIOM
  Person=unionOf((:Man,Woman:))
```

```
TallMan_subClassOf_Person: THEOREM
  subClassOf?(TallMan,Person)
```

The main steps of this proof are to prove separately `subClassOf?(TallMan,Man)` and `subClassOf?(Man,Person)`. Then the simple subsumption reasoning can finish proving the theorem. The above two goals can be proved by the application of two user defined theorems relating *intersectionOf* and *unionOf* to *subClassOf*, respectively.

Instantiation Reasoning. Instantiation reasoning asserts that one resource is or is not an instance of a class. Some SW reasoning tools such as FaCT are designed to only support concept-level reasoning. Hence reasoning at the instance-level cannot be performed by these tools. We demonstrate through an example that PVS supports instance-level reasoning.

In the example ontology, we defined an individual called *Santa*, who can move by both walking and flying, by the following axioms.

```
Santa_moves_walk_ax: AXIOM moves(Santa,walk)
Santa_moves_fly_ax: AXIOM moves(Santa,fly)
```

We want to prove that *Santa* is not an instance of the class *Person*. By stating the facts that all instances of the *Person* class can move only by walk, that the individual *Santa* can fly, and that *walk* and *fly* are disjoint, we can finish the proof with a `(grind)` command.

Table 1. The Model of Scheduling Tasks

```
:Agent a owl:Class.
:Task a owl:Class.
:TimePoint a owl:Class.
:Data a owl:Class.
:relatesTo a owl:TransitiveProperty;
  rdfs:domain Task;
  rdfs:range Task;
:assignedTo a owl:ObjectProperty;
  rdfs:domain Task;
  rdfs:range Agent.
:starts a owl:ObjectProperty;
  rdfs:domain Task;
  rdfs:range TimePoint.
:ends a owl:ObjectProperty;
  rdfs:domain Task;
  rdfs:range TimePoint.
:precedes a owl:TransitiveProperty;
  rdfs:domain TimePoint;
  rdfs:range TimePoint.
:overlaps a owl:ObjectProperty;
  rdfs:domain Task;
  rdfs:range Task.
:uses a owl:ObjectProperty;
  rdfs:domain Task;
  rdfs:range Data.
```

```
:a1 a Agent.
:a2 a Agent.
:t1 a Task;
  :starts :tp1;
  :ends :tp3;
  :assignedTo :a1.
:t2 a Task;
  :starts :tp2;
  :ends :tp4;
  :uses :d2;
  :assignedTo :a2.
:t3 a Task;
  :starts :tp4;
  :ends :tp5;
  :relatesTo :t1;
  :uses :d1;
  :assignedTo :a2.
```

```
:tp1 a TimePoint;
  :precedes :tp2.
:tp2 a TimePoint;
  :precedes :tp3.
:tp3 a TimePoint;
  :precedes :tp4.
:tp4 a TimePoint;
  :precedes :tp5.
:tp5 a TimePoint.
:d1 a Data.
:d2 a Data.
```

5.2 Checking ORL and Beyond

The above examples demonstrate PVS's power of performing consistency, subsumption and instantiation reasoning about OWL ontologies with certain degree of automation. Now we shall illustrate that PVS can reason about ORL and more complex properties that even ORL cannot capture.

ORL Reasoning. As stated earlier, one important reason of OWL expressive limitation is that while the language contains a rich set of class constructors, very little can be said about properties. Even simple composition of two properties is impossible to represent. It is for this reason that ORL is proposed. Here we demonstrate how PVS can act as a reasoner to support ORL.

We illustrate our idea with an example ontology about scheduling agents for different tasks, which is represented in n3 [3] syntax below in Table 1. The main reasoning task is, given a schedule and a set of constraints, to determine whether the schedule violates the constraints. Informally, there is a set of tasks and a set of agents. Any task can be assigned to any agent. There is also a set of discrete time points and a set of data. A time point may precede another. Each task starts and ends at a particular time point and may possibly use a piece of data. A task could relate to another task. Some tasks may overlap with some other task(s).

Four rules capture the requirements of the system. The first one states that an agent cannot be assigned to two overlapping tasks. The transformed PVS theorem is given on the right.

$$Task(t1) \wedge Task(t2) \wedge Agent(a1) \wedge Agent(a2) \wedge assignedTo(t1, a1) \wedge assignedTo(t2, a2) \wedge overlaps(t1, t2) \rightarrow differentFrom(a1, a2)$$

```
rule_1: AXIOM FORALL(t1,t2,a1,a2 : RESOURCE):
  member(t1,instances(Task)) AND
  member(t2,instances(Task)) AND
  member(a1,instances(Agent)) AND
  member(a2,instances(Agent)) AND
  assignedTo(t1,a1) AND assignedTo(t2,a2)
  AND overlaps(t1,t2)
    IMPLIES
  differentFrom?(a1,a2)
```

Since ORL rules transformation to PVS is straightforward, as we previously mentioned in Section 4, we will omit the PVS version of the following rules.

The second rule requires that related tasks must be assigned to the same agent.

$$Task(t1) \wedge Task(t2) \wedge Agent(a1) \wedge Agent(a2) \wedge assignedTo(t1, a1) \wedge assignedTo(t2, a2) \wedge relatesTo(t1, t2) \rightarrow sameAs?(a1, a2)$$

The third rule requires that any two overlapping tasks cannot use the same piece of data.

$$Task(t1) \wedge Task(t2) \wedge Data(d1) \wedge Data(d2) \wedge uses(t1, d1) \wedge uses(t2, d2) \wedge overlaps(t1, t2) \rightarrow differentFrom?(d1, d2)$$

The last rule defines when two tasks are overlapping - when one task that starts earlier ends after the other task starts.

$$Task(t1) \wedge Task(t2) \wedge TimePoint(tp1) \wedge TimePoint(tp2) \wedge TimePoint(tp3) \wedge TimePoint(tp4) \wedge starts(t1, tp1) \wedge ends(t1, tp2) \wedge starts(t2, tp3) \wedge ends(t2, tp4) \wedge precedes(tp1, tp3) \wedge precedes(tp3, tp2) \rightarrow overlaps(t1, t2)$$

To prove that the schedule violates some of the constraints, we simply prove the following PVS theorem: `violateConstraint: theorem FALSE`.

A proof strategy is intended to capture patterns of inference steps. A defined proof rule is a strategy that is applied in a single atomic step so that only the final effect of the strategy is visible and the intermediate steps are hidden from the user. We define a number of proof strategies, such as `(installTimePoint)`, `(installData)`, `(installAgent)`, etc., each of which introduces all the axioms one by one of a particular class. The following strategy introduces to PVS all facts related to all the time points.

```
(defstep installTimePoint ()
  (then
    (lemma "tp1_instanceOf_ax")
    (lemma "tp1_precedes_ax")
    (lemma "tp2_instanceOf_ax")
    (lemma "tp2_precedes_ax")
```

```
    (lemma "tp3_instanceOf_ax")
    (lemma "tp3_precedes_ax")
    (lemma "tp4_instanceOf_ax")
    (lemma "tp4_precedes_ax")
    (lemma "tp5_instanceOf_ax")
  )
  "Installing all axioms of TimePoint"
  "Installing all axioms of TimePoint"
)
```

Then we also define a strategy which finds and installs the transitive closure of the property `precedes`, i.e., the relative temporal order of all pairs of time points, as follows. This is needed for determining instances of the `overlaps` property later.

```
(defstep installAllPrecedes ()
  (then
    (lemma "precedes_transitive_ax")
    (rewrite "transitiveProperty?")
    (try (forward-chain -1) (installAllPrecedes) (delete -1))
  )
  "Finding and installing all precedes property instances"
  "Finding and installing all precedes property instances"
)
```

Basically this strategy repeatedly forward-chains the `precedes_transitive_ax` axiom until there is no more effect. Similarly, we find all instances of the property `relatesTo` by using the strategy `installAllRelatesTo` (not shown here).

Now we apply the rules. First, we apply the fourth rule to discover all instances of the property `overlaps` by using the strategy `installAllOverlaps` below.

```
(defstep installAllOverlaps ()
  (then
    (lemma "rule_4")
    (try (forward-chain -1) (installAllOverlaps) (delete -1))
  )
  "Finding and installing all overlaps property instances"
  "Finding and installing all overlaps property instances"
)
```

Then we can apply the other three rules one by one by using strategies similarly.

We apply the `(grind)` command, which proves the theorem. It means that the schedule cannot satisfy the conjunction of all constraints. A closer look at the ontology discovers that tasks $t1$ and $t2$ are related and yet overlapping. This reasoning technique becomes more important when the ontology contains more classes and more complicated properties.

Reasoning Beyond ORL. One example that OWL & ORL cannot deal with is the concrete domains: it can only make assertions about linear (in)equalities of cardinalities of property instances over integer. PVS, on the other hand, can perform basic arithmetic operations and comparisons, which we believe could improve the proof power beyond SW.

We illustrate the idea with the same schedule example. In the previous section, we model time as discrete time points and their temporal relationship as

an abstract *precedes* property. Now we can use the natural number domain to model time. Correspondingly, the *starts* and *ends* properties would have to be refined into functions from *Task* to natural number. Then the *overlaps* property could be refined as follows.

```
overlaps_ax: theorem
  FORALL (t1,t2:INDIVIDUAL):
    member(t1,instances(Task)) AND
    member(t2,instances(Task)) AND
    starts(t1) < starts(t2) AND starts(t2) < ends(t1)
      IMPLIES
    overlaps(t1,t2)
```

The above is just a simple example property that ORL cannot specify. If more constraints are to be put into the ontology, such as deadline for the whole schedule (which requires addition over numbers) or axioms other than $C(x)$, $P(x, y)$, $sameAs(x, y)$ and $differentFrom(x, y)$ are to be put into rules, more interesting properties would arise, which are also inexpressible in ORL.

6 Conclusion

Ensuring the correctness of shared ontologies is an important task in ontology development as inconsistent ontologies may lead agents to draw erroneous conclusions. In our previous works [8, 20], we have attempted to use a combination of SW and formal methods tools to reason about DAML+OIL/RDF ontologies. We used Alloy Analyzer (AA) [13], Z/EVES [18], RACER and OilEd [1] in combination to check for properties of interest. Some properties are beyond the modeling power of DAML+OIL. In this approach, the various tools were used in a complementary way such that a balance of automation and expressiveness is achieved. Moreover, the source of ontological errors can be traced in AA.

There are a few drawbacks to this approach. Firstly, AA does not scale up very well and secondly, Z/EVES works interactively, as PVS theorem prover does.

One part of the future works is to enhance the proof support for OWL and ORL. The PVS reasoner will be more effective if the semantics include not only essential functions but also sufficient supporting lemmas and theorems that makes proof of trivial goals more automated.

PVS is a generic theorem prover. As a result, it lacks complete automation. Hence, another part of the future works is to reduce user interactions as much as possible so that the reasoning procedure can be more efficient. This can be achieved by developing more advanced proof strategies.

Model-checking capabilities used for automatically verifying temporal properties of finite-state systems have recently been integrated into PVS. Hence PVS could be used to model and reason behaviors of SW services such as DAML-S [6].

In conclusion, we presented the PVS semantics for the Semantic Web ontology language OWL and its proposed extension ORL, the transformation process from OWL ontologies to PVS specifications, and our approach of using PVS theorem prover to reason ontology-related properties, sometimes beyond the modeling

capabilities of ORL. Some of the advanced features such as proof strategy are incorporated in our approach.

ORL is a newly proposed ontology language. To our knowledge, so far there is no existing reasoning system that can support ORL prior to this work. The high expressiveness of PVS language, its highly tunable proof strategies and similarity between PVS sequents and ORL rules make PVS a natural reasoner for ORL.

Acknowledgements

This work is supported by the DIRP Grant *"Formal Design Methods and DAML"* from Defense Science & Technology Agency (DSTA) Singapore. We would also like to thank Hai Wang, who directed our attention to ORL and provided valuable comments on an earlier draft of this paper.

References

1. Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: a reasonable ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, number 2174 in Lecture Notes in Computer Science, pages 396–408, Vienna, September 2001. Springer-Verlag.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.
3. Tim Berners-Lee. Notation 3 – Ideas about Web architecture. http://www.w3.org/DesignIssues/Notation3.html, 1998.
4. D. Brickley and R.V. Guha (editors). Resource description framework (rdf) schema specification 1.0. http://www.w3.org/TR/rdf-schema/, February 2004.
5. J. Broekstra, M. Klein, S. Decker, D. Fensel, and I. Horrocks. Adding formal semantics to the web: building on top of rdf schema. In *ECDL Workshop on the Semantic Web: Models, Architectures and Management*, 2000.
6. M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. Daml service. http://www.daml.org/services/daml-s/2001/05/.
7. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein (editors). OWL Web Ontology Language 1.0 Reference. http://www.w3.org/TR/owl-ref/, 2002.
8. J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. A combined approach to checking web ontologies. In *Proceedings of 13th World Wide Web Conference (WWW'04)*, pages 714–722, New York, USA, May 2004.
9. 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.
10. Volker Haarslev and Ralf Möller. *RACER User's Guide and Reference Manual: Version 1.7.6*, December 2002.
11. I. Horrocks. The FaCT system. *Tableaux'98, LNCS*, 1397:307–312, 1998.

12. Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731, New York, USA, May 2004. ACM. http://www.cs.man.ac.uk/~horrocks/DAML/Rules/.
13. D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: the Alloy Constraint Analyzer. In *The 22nd International Conference on Software Engineering (ICSE'00)*, pages 730–733, Limerick, Ireland, June 2000. ACM Press.
14. F. Manola and E. Miller (editors). RDF Primer. http://www.w3.org/TR/rdf-primer/, February 2004.
15. Daniele Nardi and Ronald J. Brachman. An introduction to description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors, *The description logic handbook: theory, implementation, and applications*, pages 1–40. Cambridge University Press, 2003.
16. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.
17. S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, December 2001.
18. M. Saaltink. The Z/EVES system. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: Z Formal Specification Notation*, volume 1212 of *Lect. Notes in Comput. Sci.*, pages 72–85. Springer-Verlag, 1997.
19. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks (editors). Reference description of the DAML+OIL ontology markup language. Contributors: T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, L. A. Stein, ..., March, 2001.
20. Hai Wang. *Semantic Web and Formal Design Methods*. PhD thesis, National University of Singapore, 2004.
21. World Wide Web Consortium (W3C). OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, March 2003.
22. World Wide Web Consortium (W3C). Web Ontology Language (OWL) Use Cases and Requirements. http://www.w3.org/TR/webont-req/, March 2003.