CS6214 Project Final Report

Feng Yuzhang

School of Computing
National University of Singapore
`fengyz@comp.nus.edu.sg`

1 Introduction

The World Wide Web has served us for more than a decade. However, to reach its full potential, the Web must evolve into Semantic Web [1] - a universal medium for sharing and processing data not only by people, but also by computers. Proposed by the inventor of the Web, Semantic Web is now an initiative of the international standardization body, the World Wide Web Consortium (W3C), which aims to develop it into the next generation of the Web.

In Semantic Web, understanding of data is achieved with ontologies: taxonomies that define and inter-relate data. Ontologies are expressed in terms of ontology languages, the building blocks of Semantic Web. The Web Ontology Language (OWL) [3], now a recommendation by W3C, defines the basic vocabulary for describing data on the web and is a layer on which Web Services can be developed.

In order for computers to share, process and integrate data correctly without human supervision, ontologies must be carefully designed, integrated and validated before deployment. This is where software verification techniques and formal methods become useful and important.

Formal methods are an active research area focusing on rigorously modelling and verifying software and hardware systems using mathematical languages. A number of formal notations, such as **Z** [11], and industry-strength verification and development tools have been built and widely used.

It is believed that Semantic Web can be a new application domain for formal methods, especially for theorem provers. PVS [8] is an interactive theorem prover with an expressive specification language. In this project, I apply PVS to reason about properties in an ontology expressed in OWL. The following techniques are employed. First I define, according the abstract syntax and semantics of OWL [9], the PVS semantics for OWL. Then I develop a transformation tool which takes in an OWL ontology and outputs a PVS specification. The ontology properties to be verified are specified in another PVS theory which imports the PVS specification of the ontology and are proved in the PVS theorem prover.

The remainder of the report is organized as follows. In Section 2, I will briefly discuss the PVS semantics for OWL and the proof supports. In Section 3, I will describe the transformation tool. Section 4 introduces the ontology used in this project. Various types of reasoning are discussed and demonstrated in Section 5. The last section contains the conclusion.

2 PVS Semantics for OWL

In order to use PVS to verify and reason OWL ontologies, it is necessary to define the PVS semantics for OWL. This semantic model forms the reasoning environment for verification using PVS theorem prover and is defined in accordance with the OWL abstract syntax and semantics [9]. In this section, I only present a PVS specification for a subset of OWL language primitives. The complete model can be found in Appendix A. The semantics are defined from basic concepts to complex class restrictions in a bottom-up fashion.

Basic Concepts

Everything in Semantic Web is built from a *Resource*. So I model it by defining a non-empty type (denoted by the plus sign) in PVS.

```
RESOURCE: TYPE+
```

In OWL Full the universe of individuals consists of all resources. Thus I 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 I model *Class* as a sub-type of *Resource* and define a function *instances* that maps a class to a set of individuals.

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

Each property in OWL relates resources to resources. So I 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.

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 modelled as a boolean function of two properties.

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

Class & Property

Class restrictions are also possible. 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 it is modelled as a function from a property p and a class $c2$ to a class $c1$. Its meaning is specified as an axiom `allValuesFrom_ax` 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)))))
```

Proof Support for PVS

To make the proving process of PVS more automated, a set of theorems is also defined. They aim to hide certain amount of underlying model from the verification and reasoning and to achieve abstraction and reuse. 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 another 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 class is also an instance of any super class of the class.

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

The set of theorems is constructed in a incremental fashion. It started as a small set containing only simple theorems, such as the `subClassOf_transitive` theorem, which are believed to be useful. During the proof process, it is realized that more complex theorems are necessary. So new theorems are defined and proved for reuse. For example, the following theorem is useful for subsumption reasoning and is added to the theorem set.

```
someValuesFrom_subClassOf: THEOREM
    FORALL (p:PROPERTY),(c1,c2,c3,c4:CLASS):
        someValuesFrom(p,c1) = c2 AND someValuesFrom(p,c3) = c4 AND subClassOf?(c1,c3)
        IMPLIES
        subClassOf?(c2,c4)
```

3 Transformation from OWL to PVS

We have developed a tool in Java to automatically transform OWL ontologies into PVS specifications. The transformation tool imports some Java packages developed by HP research lab, the Jena Framework 2 [4]. Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF [6], RDFS [2] and OWL. It is able to extract and iterate through elements such as classes, properties and individuals of an ontology. Then our tool translates them into PVS specifications according to their structures in the ontology without the introduction of new anonymous class.

For example, the following ontology fragment defines a class *MilitaryTask* and specifies some of its properties. The transformed PVS fragment is shown beneath the ontology.

```
<owl:Class rdf:about="
  http://www.dso.org.sg/PlanOntology/Ontology/DAML/mil_proc_ont.daml#MilitaryTask">
  <rdfs:comment><![CDATA[A military task is an independent entity that is
    assigned to a particular unit to execute.]]>
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:about=
      "http://www.dso.org.sg/PlanOntology/Ontology/DAML/mil_proc_ont.daml#MilitaryProcess"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource=
        "http://www.dso.org.sg/PlanOntology/Ontology/DAML/mil_proc_ont.daml
          #MilitaryTask::assignedTo"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
        1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
MilitaryTask: CLASS
MilitaryTask_subClassOf_ax_1: AXIOM subClassOf(MilitaryTask,MilitaryProcess)
MilitaryTask_subClassOf_ax_2: AXIOM subClassOf(MilitaryTask,cardinality(assignedTo,1))
```

The tool developed is also able to transform instance ontologies into PVS specifications. For example, the following shows an OWL instance ontology fragment and the corresponding PVS specification.

```
<rdf:Description rdf:about="
  http://www.dso.org.sg/PlanOntology/Ontology/DAML/unit_ech_ont.daml#SQN">
  <rdfs:comment><![CDATA[Squadron. A squadron is made up of a number of
    flights. A squadron is equivalent to a battalion.]]></rdfs:comment>
  <rdf:type>
    <owl:Class rdf:about="
      http://www.dso.org.sg/PlanOntology/Ontology/DAML/unit_ech_ont.daml#MilitaryEchelon"/>
  </rdf:type>
  <ns0:subEchelon rdf:resource="
    http://www.dso.org.sg/PlanOntology/Ontology/DAML/unit_ech_ont.daml#BDE"/>
  <ns0:subEchelon rdf:resource="
    http://www.dso.org.sg/PlanOntology/Ontology/DAML/unit_ech_ont.daml#FLOTILLA"/>
</rdf:Description>
```

```
SQN: INDIVIDUAL
SQN_MilitaryEchelon_ax: AXIOM member(SQN,instanceOf(MilitaryEchelon))
SQN_subEchelon_BDE_ax: AXIOM subEchelon(SQN,BDE)
SQN_subEchelon_FLOTILLA_ax: AXIOM subEchelon(SQN,FLOTILLA)
```

4 Military Plan Ontology

DSO National Laboratories (DSO) developed a OWL military plan ontology [5], defining concepts in the military domain, including military organizations, specialities, geographic features, etc. For example, the class `MilitaryTask`, a subclass of `MilitaryProcess`, is defined as follows.

```
<owl:Class rdf:about="
  http://www.dso.org.sg/PlanOntology/Ontology/DAML/mil_proc_ont.daml#MilitaryTask">
  <rdfs:comment><![CDATA[A military task is an independent entity that is
    assigned to a particular unit to execute.]]>
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:about=
      "http://www.dso.org.sg/PlanOntology/Ontology/DAML/mil_proc_ont.daml#MilitaryProcess"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource=
        "http://www.dso.org.sg/PlanOntology/Ontology/DAML/mil_proc_ont.daml
          #MilitaryTask::assignedTo"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
        1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

A number of plan instances of this ontology were also generated from plain text by an information extraction (IE) engine developed by DSO. Military plans are typically prepared as both graphical overlays and textual documents detailing the plans. IE is used as the first part of a process to transform the textual documents into ontological data. A typical IE workflow consists of word segmentation & stemming, POS (part of speech) tagging, Named Entity recognition, etc. With all information gathered from the various steps, the IE engine then fills the slots in pre-defined templates, which is subsequently transformed into a RDF document. Generally speaking, an instance ontology is made up of the following four main ingredients.

– A set of military operations and tasks, defining their types, phases and the logic order.
– A set of military units, which are the participants of the military operations and tasks,
– A set of geographic locations, where such operations take place and
– A set of time points for constraining the timing of such operations.

5 Ontology Reasoning Using PVS

In this section, we demonstrate how PVS can be used to check ontology-related properties and the various types of ontology reasoning. Standard SW reasoning includes three categories, namely inconsistency checking, subsumption reasoning and instantiation reasoning. The following sections give examples of the various types of reasoning. The proof trees of the example theorems can be found in Appendix B.

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 [7].

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

```
PrepareDemolition_MilitaryTask : CLASS
MilitaryTask,EngineerUnit,ArtilleryFiringUnit : CLASS
assignedTo:O_PROPERTY

PrepareDemolition_MilitaryTask_subClassOf_ax_1: AXIOM
    subClassOf?(PrepareDemolition_MilitaryTask,MilitaryTask)
PrepareDemolition_MilitaryTask_subClassOf_ax_2: AXIOM
    subClassOf?(PrepareDemolition_MilitaryTask,allValuesFrom(assignedTo,EngineerUnit))
PrepareDemolition_MilitaryTask_subClassOf_ax_3: AXIOM
    subClassOf?(PrepareDemolition_MilitaryTask,allValuesFrom(assignedTo,ArtilleryFiringUnit))
MilitaryTask_subClassOf_ax_2: AXIOM
    subClassOf?(MilitaryTask,someValuesFrom(assignedTo,ModernMilitaryUnit))
ArtilleryFiringUnit_disjointWith_ax_1: AXIOM
    disjointWith?(ArtilleryFiringUnit,EngineerUnit)
```

It is suspected that there is an inconsistency in the class *PrepareDemolition_MilitaryTask*. To prove that, it is equivalent to prove the following theorem, which means that the class *PrepareDemolition_MilitaryTask* does not admit any individual.

```
PDMT_inconsistent : THEOREM
    (EXISTS (i:INDIVIDUAL): member(i,instances(PrepareDemolition_MilitaryTask))) IMPLIES FALSE
```

Figure 1 in Appendix B is the proof tree of the theorem PDMT_inconsistent.

Subsumption Reasoning

The task of subsumption reasoning is to infer that an OWL class is a sub-class of another. There are many ways of conducting subsumption reasoning. The simplest and most essential way is by using the transitivity of the *subClassOf* relation. Another way of doing subsumption reasoning is by using inter-class relationships such as *intersectionOf* and *unionOf*. More advanced and complex subsumption reasoning can be based on class restrictions such as *someValuesFrom*. Each of the above methods are illustrated with a user-defined theorem which can be instantiated during proof.

– Transitivity of *subClassOf*

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

Figure 2 in Appendix B is the proof tree of the theorem subClassOf_transitive.

– Inter-class relationships

```
unionOf_subClassOf: THEOREM
    FORALL (c:CLASS),(lc:list[CLASS]):
        unionOf(lc)=c IMPLIES
            FORALL (sc:CLASS):
                member(sc,lc) IMPLIES subClassOf?(sc,c)

intersectionOf_subClassOf: THEOREM
    FORALL (c:CLASS),(lc:list[CLASS]):
        intersectionOf(lc)=c IMPLIES
            FORALL (sc:CLASS):
                member(sc,lc) IMPLIES subClassOf?(c,sc)
```

Figure 3 and 4 in Appendix B are the proof trees of the theorem unionOf_subClassOf and intersectionOf_subClassOf respectively.

– Class restrictions

```
someValuesFrom_subClassOf: THEOREM
    FORALL (p:PROPERTY),(c1,c2,c3,c4:CLASS):
        someValuesFrom(p,c1)=c2 AND someValuesFrom(p,c3)=c4 AND subClassOf?(c1,c3)
        IMPLIES
        subClassOf?(c2,c4)
```

Figure 5 in Appendix B is the proof tree of the theorem someValuesFrom_subClassOf.

Instantiation Reasoning

Instantiation reasoning asserts that one resource is or is not an instance of a class. In the example ontology, there is an individual called *CDF*3 (for Civil Defence Force 3). I want to prove that *CDF*3 is not an instance of the class *ModernMilitaryUnit*. The proof goal is specified in the following theorem.

```
CDF3_Not_ModernMilitaryUnit: THEOREM
    NOT member(CDF3,instances(ModernMilitaryUnit))
```

Figure 6 in Appendix B is the proof tree of the theorem CDF3_Not_ModernMilitaryUnit.

6 Conclusion

This project uses PVS theorem prover to prove important ontology properties in a case study, the military plan ontology. There are three major achievements of the project. First I defined the PVS semantics for ontology language OWL. Second, A transformation tool is developed to perform automatic transformation of OWL and RDF ontologies into PVS specifications. Third I performed inconsistency checking, subsumption reasoning and instantiation reasoning with the aid of PVS theorem prover.

One possible future work is to employ model-checkers to verify the behaviours of Semantic Web services [10] as they are dynamic in nature.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.
2. D. Brickley and R.V. Guha (editors). Resource description framework (rdf) schema specification 1.0. URL: http://www.w3.org/TR/rdf-schema/, February 2004.
3. 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. URL: http://www.w3.org/TR/owl-ref/, 2002.
4. HP Labs. Jena 2 - A Semantic Web Framework. URL: http://www.hpl.hp.com/semweb/jena2.htm.
5. C. H. Lee. Phase I Report for Plan Ontology. DSO National Labs, Singapore, 2002.
6. F. Manola and E. Miller (editors). RDF Primer. URL: http://www.w3.org/TR/rdf-primer/, February 2004.
7. 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.
8. 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.
9. P. F. Patel-Schneider, P. Hayes, and I. Horrocks (editors). OWL Web Ontology Semantics and Abstract Syntax. URL: http://www.w3.org/TR/2004/REC-owl-semantics-20040210/, 2004.
10. The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. URL: http://www.daml.org/services/owl-s/, 2004.
11. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall International, 1996.

A Complete PVS Semantics for OWL

This section provide the complete PVS semantics for OWL which is described briefly earlier in Section 2.

```
owl_full : THEORY

BEGIN

    IMPORTING finite_sets

    RESOURCE: TYPE+
    INDIVIDUAL: TYPE+ = RESOURCE
    DATAVALUE: TYPE+ FROM INDIVIDUAL
    CLASS: TYPE+ FROM RESOURCE
    PROPERTY: TYPE = pred[[RESOURCE,RESOURCE]]
    O_PROPERTY: TYPE+ FROM PROPERTY
    D_PROPERTY: TYPE+ FROM PROPERTY
    instances: [CLASS -> set[INDIVIDUAL]]

    thing: CLASS
    thing_ax: AXIOM FORALL (i:INDIVIDUAL): member(i,instances(thing))
```

```
nothing: CLASS
nothing_ax: AXIOM FORALL (i:INDIVIDUAL): NOT member(i,instances(nothing))

oneOf : [set[INDIVIDUAL] -> CLASS]
oneOf_ax : AXIOM FORALL (si:set[INDIVIDUAL]),(c:CLASS):
    (oneOf(si) = c IMPLIES instances(c) = si)

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

someValuesFrom : [PROPERTY, CLASS -> CLASS]
someValuesFrom_ax : AXIOM
    FORALL (c1,c2:CLASS),(p:PROPERTY):
        (c1 = someValuesFrom(p,c2) IMPLIES
            FORALL (i1:INDIVIDUAL):
                (member(i1,instances(c1)) IFF
                    EXISTS (i2:INDIVIDUAL):
                        (member(i2,instances(c2)) AND p(i1,i2))))

hasValue : [PROPERTY, INDIVIDUAL -> CLASS]
hasValue_ax : AXIOM
    FORALL (c:CLASS),(p:PROPERTY),(i:INDIVIDUAL):
        (c = hasValue(p,i) IMPLIES
            FORALL (i1:INDIVIDUAL):
                (member(i1,instances(c)) IFF p(i1,i)))

maxCardinality : [PROPERTY, nat -> CLASS]
maxCardinality_ax : AXIOM
    FORALL (c:CLASS),(p:PROPERTY),(n:nat):
        (c = maxCardinality(p,n) IMPLIES
            FORALL (i:INDIVIDUAL):
                (member(i,instances(c)) IFF
                    EXISTS (s:finite_set[RESOURCE]):
                        (card(s) = n AND subset?(image(p,singleton(i)),s))))

minCardinality : [PROPERTY, nat -> CLASS]
minCardinality_ax : AXIOM
    FORALL (c:CLASS),(p:PROPERTY),(n:nat):
        (c = minCardinality(p,n) IMPLIES
            FORALL (i:INDIVIDUAL):
                (member(i,instances(c)) IFF
                    EXISTS (s:finite_set[RESOURCE]):
                        (card(s) = n AND subset?(s,image(p,singleton(i))))))

cardinality : [PROPERTY, nat -> CLASS]
cardinality_ax : AXIOM
    FORALL (c:CLASS),(p:PROPERTY),(n:nat):
        (c = cardinality(p,n) IMPLIES
            FORALL (i:INDIVIDUAL):
                (member(i,instances(c)) IFF
                    EXISTS (s:finite_set[RESOURCE]):
                        (card(s) = n AND s = image(p,singleton(i)))))

intersectionOf : [list[CLASS] -> CLASS]
intersectionOf_ax : AXIOM
    FORALL (lc:list[CLASS]),(c:CLASS):
        (c = intersectionOf(lc) IMPLIES
            FORALL (i:INDIVIDUAL):
                (member(i,instances(c)) IFF
                    FORALL (sc:CLASS):
                        (member(sc,lc) IMPLIES member(i,instances(sc)))))
```

```
unionOf : [list[CLASS] -> CLASS]
unionOf_ax : AXIOM
    FORALL (lc:list[CLASS]),(c:CLASS):
        (c = unionOf(lc) IMPLIES
            FORALL (i:INDIVIDUAL):
                (member(i,instances(c)) IFF
                    EXISTS (sc:CLASS):
                        (member(sc,lc) AND member(i,instances(sc)))))

complementOf : [CLASS -> CLASS]
complementOf_ax : AXIOM
    FORALL (c1,c:CLASS):
        (c = complementOf(c1) IMPLIES complement(instances(c1)) = instances(c))

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

equivalentClass?(c1,c2:CLASS): bool =
(
    instances(c1) = instances(c2)
)

disjointWith?(c1,c2:CLASS): bool =
(
    disjoint?(instances(c1),instances(c2))
)

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

domain?(p:PROPERTY,c:CLASS): bool =
(
    FORALL (i1,i2:INDIVIDUAL):
        (p(i1,i2) IMPLIES member(i1,instances(c)))
)

range?(p:PROPERTY,c:CLASS): bool =
(
    FORALL (i1,i2:INDIVIDUAL):
        (p(i1,i2) IMPLIES member(i2,instances(c)))
)

equivalentProperty?(p1,p2:PROPERTY): bool =
(
    p1 = p2
)

inverseOf?(p1,p2:PROPERTY): bool =
(
    FORALL (i1,i2:INDIVIDUAL):
        (p1(i1,i2) IFF p2(i2,i1))
)

sameAs?(i1,i2:RESOURCE): bool =
(
    i1 = i2
)

sameAs?(p1,p2:PROPERTY): bool =
(
    p1 = p2
)
```

```
    differentFrom?(i1,i2:INDIVIDUAL): bool =
    (
        NOT sameAs?(i1,i2)
    )

    allDifferent?(li:list[INDIVIDUAL]): RECURSIVE bool =
        CASES li OF
            null: TRUE,
            cons(hd, tl): NOT member(hd,tl) AND allDifferent?(tl)
        ENDCASES
        MEASURE length(li)

    transitiveProperty?(p:PROPERTY): bool =
    (
        FORALL (r1,r2,r3:RESOURCE): (p(r1,r2) AND p(r2,r3)) IMPLIES p(r1,r3)
    )

    functionalProperty?(p:PROPERTY): bool =
    (
        FORALL (r1,r2,r3:RESOURCE): (p(r1,r2) AND p(r1,r3)) IMPLIES r2=r3
    )

    symmetricProperty?(p:PROPERTY): bool =
    (
        symmetric?(p)
    )

    inverseFunctionalProperty?(p:PROPERTY): bool =
    (
        FORALL (r1,r2,r3:RESOURCE): (p(r1,r3) AND p(r2,r3)) IMPLIES r1=r2
    )

END owl_full
```

B Proof Trees

This section gives the proof trees of the theorems in Section 5

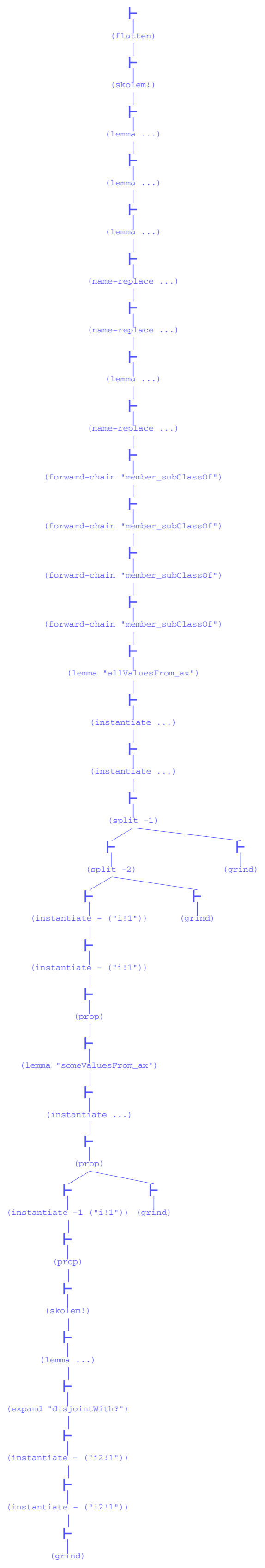

Fig. 1. Proof tree 1.

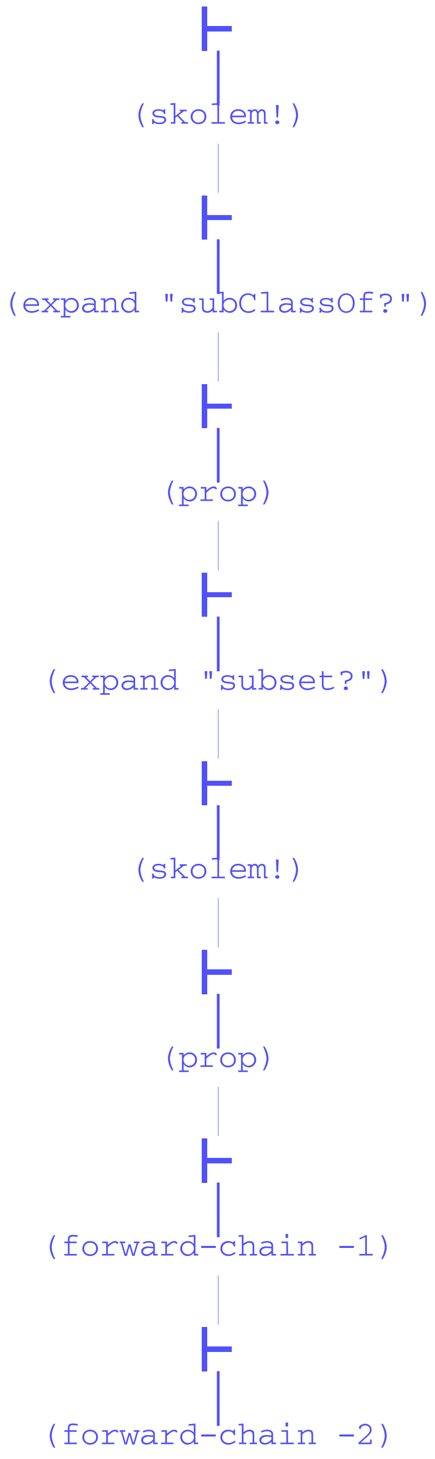

Fig. 2. Proof tree 2.

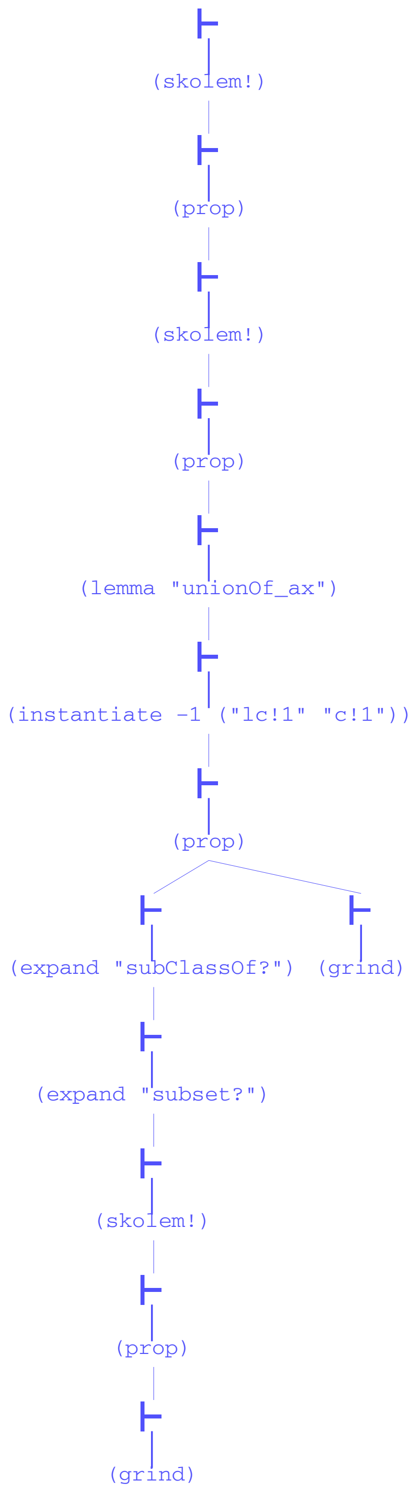

Fig. 3. Proof tree 3.

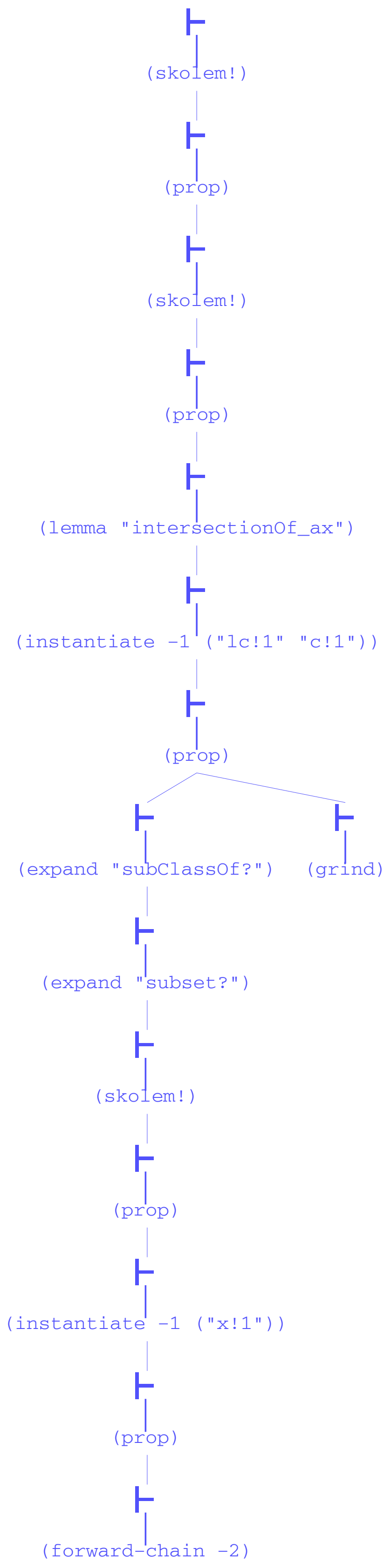

Fig. 4. Proof tree 4.

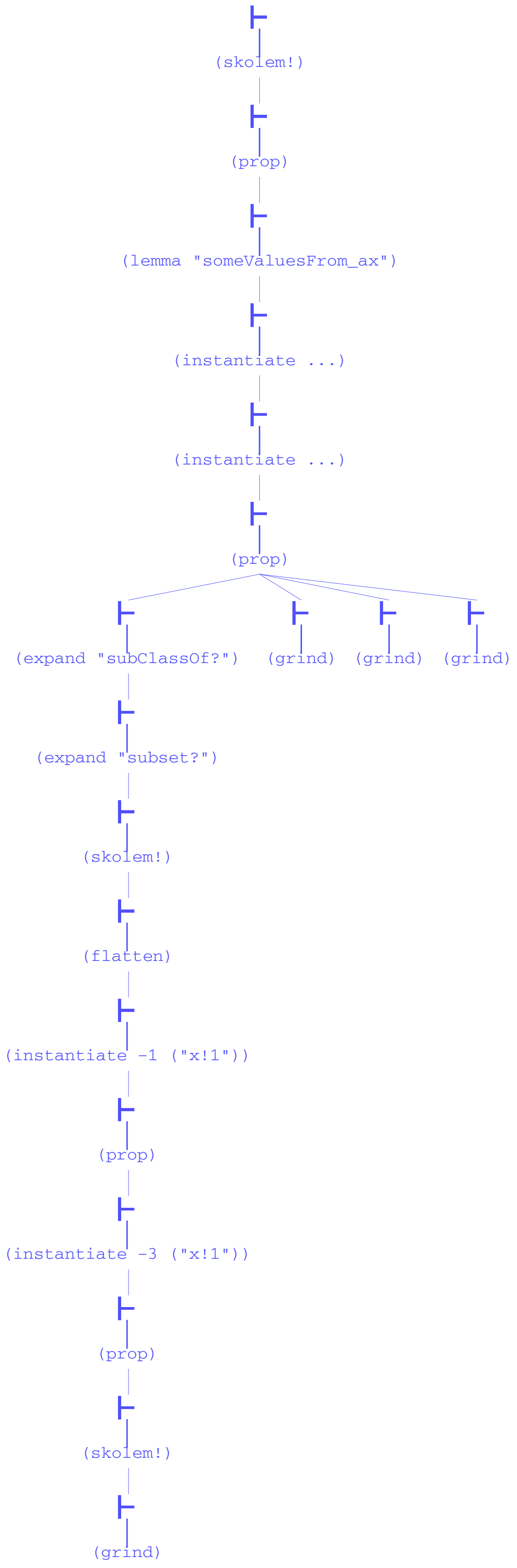

Fig. 5. Proof tree 5.

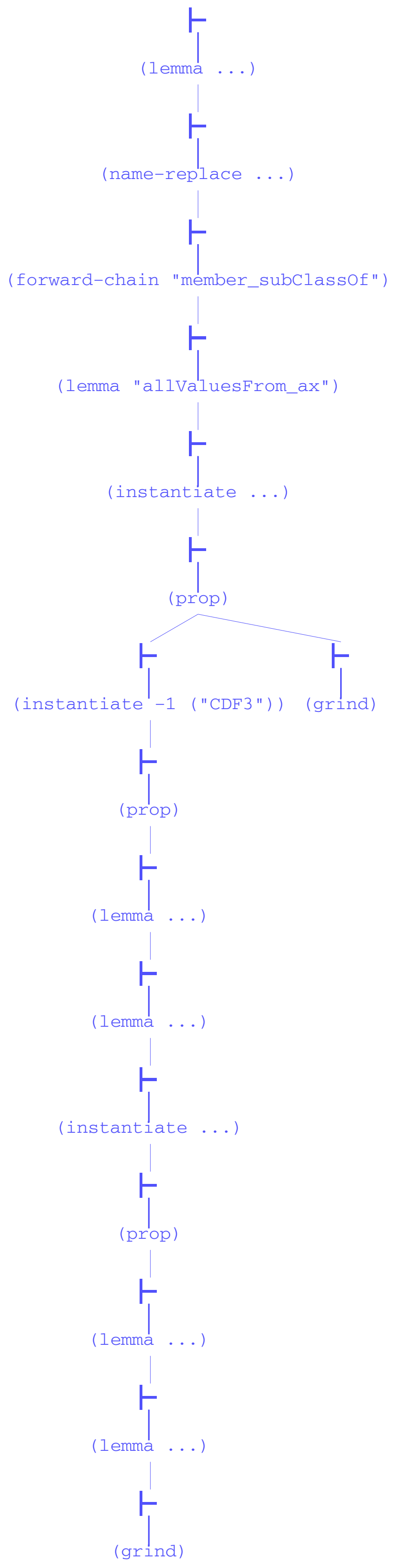

Fig. 6. Proof tree 6.