Fall 12-2010

Extending OWL with Finite Automata Constraints

Jignesh Borisa
San Jose State University

Extending OWL with Finite Automata Constraints

A Writing Project
Presented to
The Faculty of the department of Computer Science
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Jignesh Borisa
Fall 2010

SAN JOSE STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project titled

Extending OWL with Finite Automata Constraints

by

Jignesh Borisa

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science	12/01/2010
Dr. Jon Pearce, Department of Computer Science	12/01/2010
Dr. Robert Chun, Department of Computer Science	12/01/2010

ABSTRACT

Extending OWL with Finite Automata Constraints

by Jignesh Borisa

The Web Ontology Language (OWL) is a markup language for sharing and publishing data using ontologies on the Internet. It belongs to a family of knowledge representation languages for writing ontologies. Answer Set Programming (ASP) is a declarative programming approach to knowledge representation. It is oriented towards difficult search problems. In this project, we developed an extension to OWL add support for collection class constraints. These constraints come in the form of membership checks for sets where these set are computed by finite automata. We developed an inference engine for the resulting language. This engine extends the Java-based Pellet library which can reason about an OWL document.

ACKNOWLEDGEMENTS

I am grateful to my project advisor, Dr. Chris Pollett, for his suggestions and guidance. I would like to thank my committee members, Dr. Jon Pearce and Dr. Robert Chun, for their time and effort. I would also like to thank my family and friends for their supports.

Table of Contents

List of Figures

List of Tables

1 Introduction

In computer science, an ontology is used for representing knowledge about concepts in the domain and the interrelationships between those concepts. Specifically, it is used for reasoning about the properties of that domain as well as describing that domain. Ontologies and the languages representing them are used in knowledge representation areas like the semantic web, software engineering, information architecture, etc. The formal semantics of these languages can be given using Description Logics (DLs) which provide procedures to reason with the ontology knowledge. OWL is a particular language used to represent ontologies. Logic programming paradigms are also used for knowledge representation. In this project, we consider answer set programming, a particular form of logic programming. One recent extension to answer set programming is to support representing information about infinite sets. To do it, this extension supports constraints where membership in the sets can be computed by finite automata. In this extension, new types of constraints are introduced that allow for a more compact representation of problem in answer set programming. We attempted to extend OWL with feature from this new approach to answer set programming.

Let us now describe answer set programming for knowledge representation in more detail. Answer set programming is based on the stable model semantics of logic programming. The stable model semantics was proposed by Gelfond and Lifschitz [6] in 1988. It defines a declarative semantics for logic programs with negation as failure. Let P be a logic program. Let Q be a subset of variables of P. Let P^Q be the program. If the program contains clause C of P, which contains the negated variable Not A in its body such that $A \in Q$, then C is not counted. But if a body of clause contains a negated Not A such that $A \notin Q$, then Not A is not counted from the clause body. If Q is a least Herbrand model of P^Q, then Q is a stable model of P. A Herbrand

model for vocabulary V is any set of ground atoms in V. A vocabulary V consists of a set of relation constants, a set of function constants, a non-empty set of object constants and a set of variables. Marek and Remmel[1] showed that if the languages accepted by finite automata are represented by the atoms of the sets in finite base program P, and the operators involved in the program P have a certain property, then the languages accepted by finite automata are all the stable models of P. They developed extensions to the answer set programming for reasoning about infinite languages which are accepted by Deterministic Finite Automata (DFA).

The information contained in documents is not only presented to humans directly but also required to be processed by the applications. OWL is used by the applications for processing information contained in documents. OWL is used for representing the meaning of the terms and for showing relationship between those terms. This representation of terms and their interrelationship is called an ontology. OWL provides the capability to represent machine interpreted content on the web. It contains tags for describing classes and properties: relationship between classes, characteristics of properties, equality, enumerated classes, richer typing of properties, and cardinality. As an examples let Daughter be an OWL class. It contains object properties like hasFather, likes, dislikes, hasAge,etc. and data property like hasTelephone, etc. It has instances like ally, alice, sally, etc. The instances of the Daughter can have particular values for its properties like the value of hasTelephone is 314-985-8888, the value of hasAge is 19, etc. OWL allows these properties values. More specifically, Daughter class cannot support data property value as a regular expression. For example, we might want Daughter class has [314]{3}-[0-9]{3}-[0-9]{4} as value of hasTelephone property. An OWL parser would not parse this hasTelephone's value because a regular expression describes a set of strings and OWL does not allow an infinite set of possible values for a property. In our project, collections can be a set

of any real world objects like DNA sequences, telephone number Patterns, email ID patterns etc. OWL does not support collections.

The main goal of this paper is to develop some extensions to the current OWL that allows one to support collections and constraints where membership in the collection can be computed by finite automata. We also develop an inference engine to support our extension to OWL. We used the Pellet reasoner to reason about OWL document and extend it. Pellet is an open source Java-based OWL reasoner. It is used for ontology development, web service composition, and rule integration. We extended Pellet to support our extension to OWL. We have created three different inferences to reason about extended OWL document by using Pellet.

For this project, two main deliverables were produced: to develop an extension to OWL to support collection. Deliverable 1 consisted of creating and developing new tags which can make OWL capable of supporting collections. Deliverable 2 involved parsing extended OWL document and checking stable model by guessing and deriving values for the constraints. It consisted of developing an inference engine for extending Pellet. This paper describes how the extension was created and developed. The outline of this paper is as follows: Section 2 describes the tools used, Section 3 describes the preliminary work that we have done, Section 4 extends OWL to support collections and constraints, Section 5 enumerates implementation of the project and results, and Section 6 contains conclusions.

2. Tools Used

In the semantic web, information is given an explicit meaning, making it easier for machine interpretation and also to integrate information available on the web. The semantic web is built on Extensible Markup Language (XML) and Resource Description Framework (RDF). It requires ontology language for describing the meaning of terminology used in web documents. If machines are used for reasoning about these documents, the language must go beyond the basic semantics of RDF Schema. If machines are used for reasoning about these documents, we are required to use OWL. The majority of work done in this project is associated with OWL. We have used the following tools:

2.1 OWL 2.0

OWL is designed to use applications that need to process the information of content instead of just presenting information to humans [11]. OWL supports entities like class, object property, data type, data property, instances, etc. OWL also supports axioms like subClassOf, subPropertyOf, intersectionOf, etc. OWL has three sublanguages: OWL Lite, OWL DL, and OWL Full.

OWL is also used to represent the meaning of terms in vocabularies and their relationships. OWL adds more vocabulary for describing properties and classes. OWL Lite supports users who need classification hierarchy and some constraints. OWL DL supports users who want the maximum expressiveness by retaining decidability and computational completeness. OWL Full supports users who want maximum expressiveness and the syntactic freedom of RDF. OWL ontology contains a set of axioms. These axioms place constraints on sets of individuals and the types of relationships permitted between them. These axioms provide semantics by allowing additional information based on the data explicitly provided.

2.2 Pellet

Pellet is a Java-based OWL reasoner. It is open source library. It can be used in conjunction with both Jena and OWL API libraries and also provides a DIG interface. It generates OWL ontology for OWL document. Pellet provides various features like data type reasoning, ontology analysis and repair, ontology debugging, conjunctive Abox query, etc. It is basically used for application like ontology development, web service composition, and rule integration. We have used Pellet to reason about OWL document. To support our extension to OWL, we have also extended Pellet.

2.4 DOM API

DOM API is a component API of the Java API for XML processing. It allows programs to dynamically access cum update the content and structure of documents. The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of documents. We have used DOM API to parse an extended OWL document.

2.5 Smodels 2.34

Smodels is a software package used to compute stable model semantics for logic programs. Smodels can be used as a C++ library or a standalone program. Smodels extends the normal logic programs by adding new rule types like constraint rules, choice rules, and weight rules. The main front-end of smodels is lparse. We have experimented with smodels to get knowledge about constraint rules. It helped in creating rules for OWL by adding constraints. We have experimented with smodels to get an idea of how it works and handles constraints.

3 Preliminary Works

All of the initial research about the project was done during my CS297 (Preparation for writing project). During my research phase, I have learnt about all the tools required for this project and performed a few experiments using them. I have categorized my work into four deliverables: compute stable model semantics, experiment with smodels, create and reason about test OWL document and implement finite closure algorithm.

3.1 Compute stable model semantics

The stable model semantics introduced by Gelfond and Lifschitz[6] is a widely studied semantics for normal logic program. It is a tool to provide declarative semantics for logic programs with negation. This semantics is a standard approach to the meaning of negation in logic programming. This deliverable is mainly prepared to compute stable model semantics and learn answer set programming. We have written a Java Program to compute stable model semantics. Consider following examples:

Example 1:

Rules

x1:- x4.
x2: -x4,x5.
x3:- x1,x3,-x2.

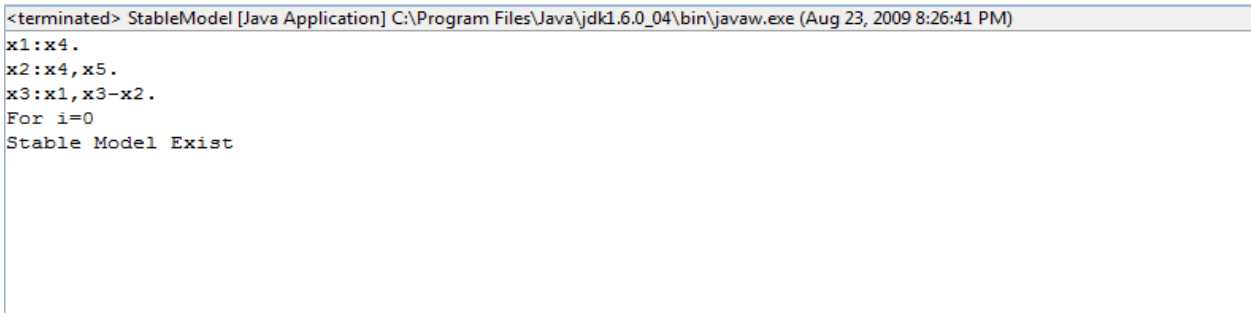

Figure 1: Compute Stable Model Semantics Example 1

Example 2

Consider the following logic program:

Rules:

x1 :- ¬ x2.
x2 :- ¬ x1.
The reduced model for the logic program is as follows:

x1:-
x2:-

I guessed for all the variables in logic program from 0 to 2^2 times. The truth table shows that guessed values for variables and checks the existence of stable model for the guessed values. The truth table is as follows:

x1	x2	Does Stable Model exist?
False	False	No
False	False	Yes
False	False	Yes
False	False	No

Table 1: Truth table for example 2

Example 3:

Consider the following logic program:

Rules:

x1:- x4,¬x2.
x2:- x4,¬x3.
x3:- ¬x2.

The reduced model for the logic program is as follows:

x1:- x4.
x2:-x4.
x3:-

x1	x2	x3	x4	Does Stable Model exist?
False	False	False	False	No
False	False	False	True	No
False	False	True	False	Yes
False	False	True	False	No
False	True	False	False	No
False	True	False	True	Yes
False	True	True	False	No
False	True	True	True	No
True	False	False	False	No
True	False	False	True	No
True	False	True	False	No
True	False	True	True	Yes
True	True	False	False	No
True	True	False	True	No
True	True	True	False	No
True	True	True	True	No

Table 2: Truth table for example 3

3.2 Experiments with smodels

We installed smodels-2.33 with lparse-1.1.1. We made changes and set objects values in the programs as per logic programs. The main goal of this deliverable is to experiment with smodels. Smodels shows the existence of stable model as output. We found stable model as an efficient program to compute stable model semantics.

```
jignesh@jignesh-laptop:~/Desktop/smodels-2.33/examples$ ./example
a :- b, c, not d, not e.
b :- a, not d.
c :- a, not b.
Stable Model:
a is not in the stable model
```

Figure 2: Smodels output

3.3 Create and reason about test OWL document

We have created a sample OWL document on computer ontology. I have learnt OWL to create sample OWL documents. Computer ontology contains OWL classes like optical+led, scroll+led, mouse, mouse+led, etc. We have used Pellet's library to reason about OWL documents. We have experimented with various inferences by using Pellet. One of the inferences was explanation. It gave subclass explanation.

```
<terminated> ExplanationExample (2) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe
Why is optical+led subclass of mouse+led?
Explanation:
   has_part range optical
   mouse+led equivalentTo mouse
                          and has_part some optical
   optical+led equivalentTo mouse
                            and has_part min 5
```

Figure 3: Output of Pellet's reasoner

In figure 3, mouse+led and optical+led are OWL classes. Pellet extracted the ranges and properties of both classes and gave explanation for the given inference.

3.4 Implement finite automata closure algorithm

We have removed negation symbol from logic program. We have added constraints to the rules. This deliverable is mainly required to compute stable model semantic by removing negative variables and adding constraints to the rules of logic program. Constraints are part of rules which contain list of variables and models. If any rule satisfies all of its constraints, it will be used for finding a stable model. For example, the logic program is as follows:

R0=0*1?

R1=1*01

x1:-|43R0;

x2:-x3|213R0,34R1;

Here, R0 and R1 are models and it contains regular expressions. 43R0 and 213R0 and 34R1 are constraints. To consider rule[0] and rule[1] for finding stable model, its constraints should be satisfied.

```
<terminated> StableModel1 [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe (
x1:|43R0;
x2:x3|213R0,34R1;
For i=0
Stable Model does not Exist
For i=1
Stable Model does not Exist
For i=2
Stable Model does not Exist
For i=3
Stable Model does not Exist
For i=4
Stable Model Exist
```

Figure 4: Finite automata closure algorithm output

4 Design and Specification

As mentioned before, OWL does not support collections. We have developed an extension to OWL to support collections and finite automata constraints. In order to develop an extension to support collection, we have created and added few tags to OWL.

4.1 CollectionClass

CollectionClass is added to OWL to support infinite sets. CollectionClass can contain one or more member classes (i.e. OWL Class). CollectionClass has data property which contains regular expression or pattern as value. CollectionClass cannot have objects.

4.1.1 CollectionClass Specification

The CollectionClass as an entity is defined by a URI. The syntax for encoding entity URIs is as follows:

datatypeURI : = URI

dataPropertyURI := URI

CollectionClassURI : = URI

The syntax for CollectionClass is as follow:

entity : = datatype | CollectionClass | dataProperty

datatype := 'Datatype' '(' datatypeURI ')'

CollectionClass : = ‘ CollectionClass’ ‘(‘ CollectionClassURI ‘)’

dataProperty : = ‘DataProperty’ ‘(‘ dataPropertyURI ‘)’

Here, CollectionClass, datatype and dataProperty are entities.

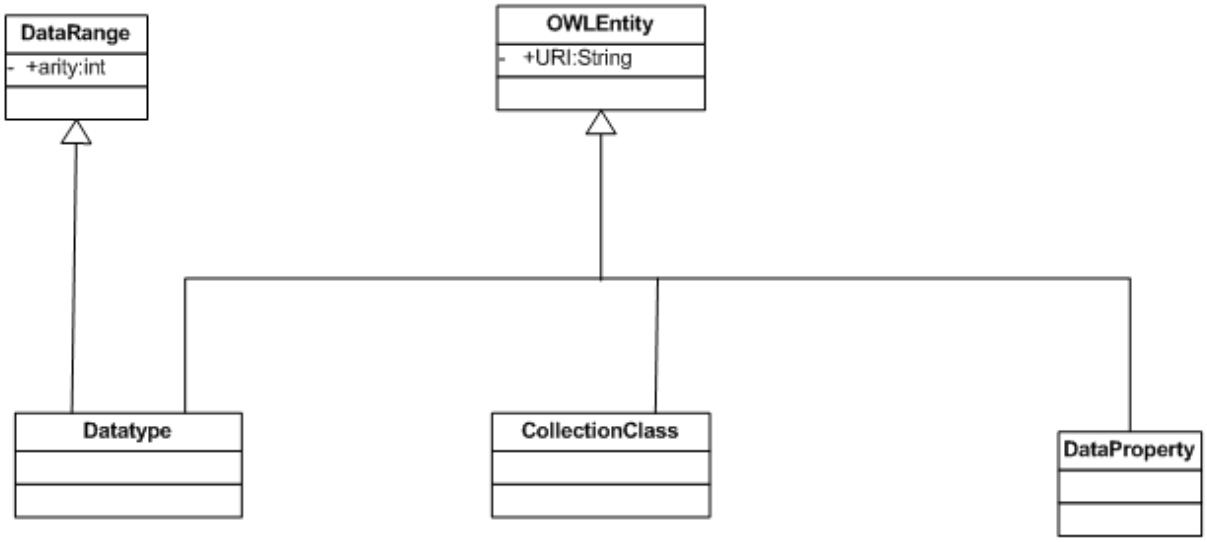

Figure 5: CollectionClass

4.2 memberClassOf

memberClassOf axiom states that one class is a member of CollectionClass. To add finite automata constraints, we added memberClassOf axiom to OWL. Any owlClass can contain one or more memberClassOf axioms. We are mapping CollectionClass property's value with the same property values of all the instances of member class. If all the instances of member class match the CollectionClass regular expression or pattern, then it can satisfy the constraint. If all of the constraints are satisfied then we can say that the document is ready for reasoning.

4.2.1 memberClassOf Specifications

The syntax of memberClassOf axiom is as follows:

owlClass : = description

CollectionClass : = description

memberClassOf : = 'MemberClassOf' '('{ annotation } owlClass CollectionClass ')'

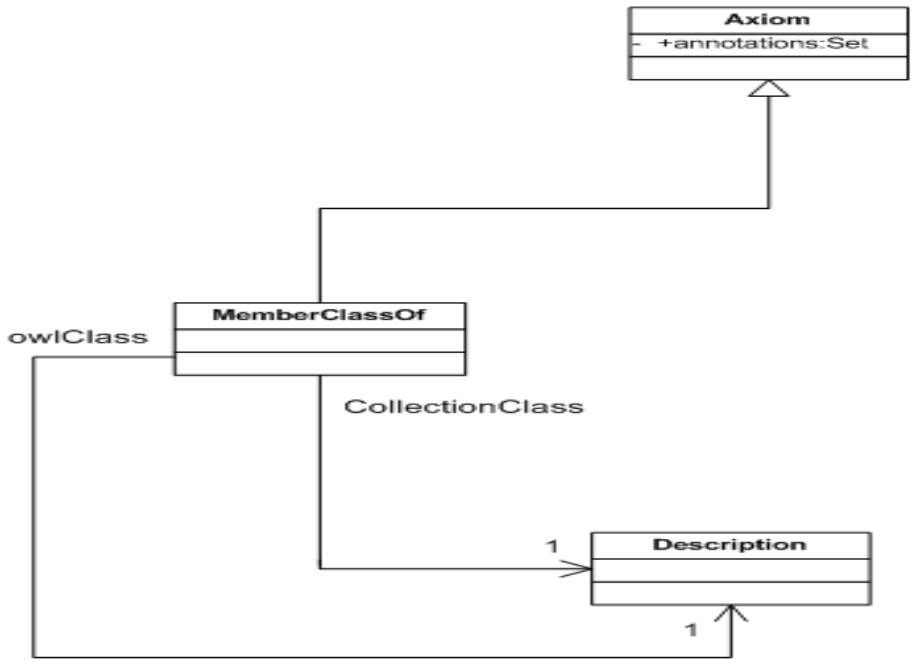

Figure 6: memberClassOf axiom

4.3 instanceOf

instanceOf axiom states that one instance is an object of the class. To add the positive variables to the rules, we added this axiom to OWL. Any owlClass can contain one or more instanceOf axioms.

4.3.1 instanceOf Specification

The syntax for instanceOf axiom is as follows:

instance : = individual

owlClass : = description

instanceOf : = 'InstanceOf' '(' { annotation } instance owlClass ')'

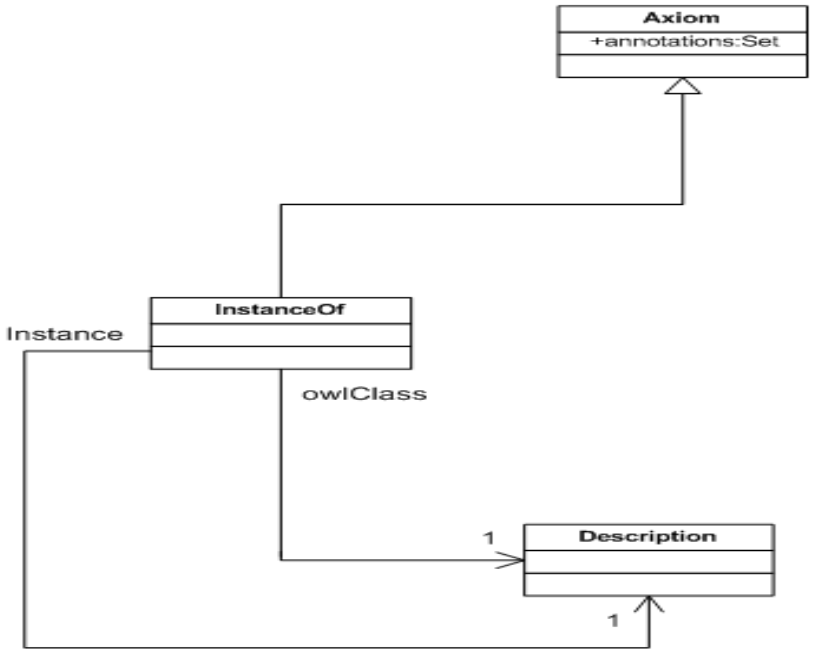

Figure 7: instanceOf Axiom

4.4 collectionClassOf

collectionClassOf axiom is used with memberClassOf to add finite automata constraint to OWL.

4.4.1 collectionClassOf Specification

The syntax for collectionClassOf axiom is as follows:

memberClass : = description

CollectionClass : = description

collectionClassOf : = 'CollectionClassof' '(' { annotation } memberClass

CollectionClass ')'

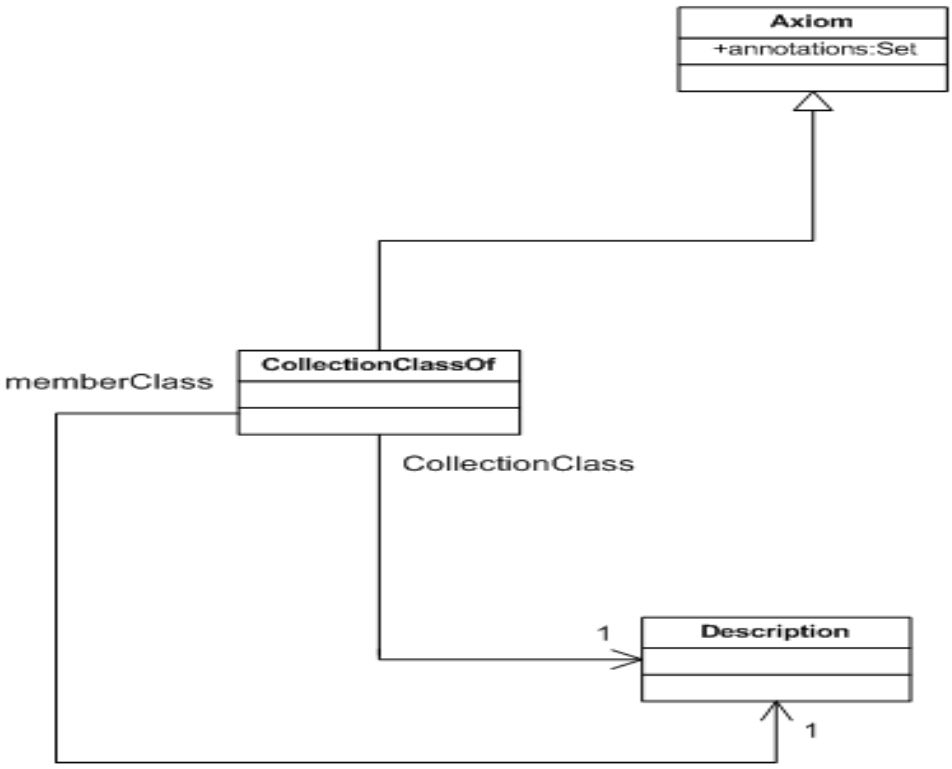

Figure 8: collectionClassOf Axiom

5 Implementation and Results

This section describes how OWL was extended to support collections. It also describes an extension to the Pellet inference engine in order to support an extended OWL document. In addition, it explains the procedure followed to use an inference engine to test the extension to OWL and also demonstrate the results obtained.

As mentioned in Section 4, we invented a few tags to support collections and create finite automata constraints. In order to extend OWL to support infinite sets, we needed to create and develop new tags that can allow OWL to support collections. To test this extension to OWL, we also required to extend the inference engine. This inference engine can reason about an extended OWL document and show results as per inferences. Therefore, the project is mainly divided into two deliverables: developing extension to OWL for supporting collections and testing the extended OWL with inference engine.

5.1 Deliverable 1: Create and Develop extension to OWL

5.1.1 Motivation

The main motivation of this deliverable was to find ways to extend OWL to support representing information about infinite sets, and allow the programs to process infinite sets in meaningful manner. It needs careful analysis of OWL. Suppose Person is an OWL class. It contains object properties like hasAge, hasFather, hasMother etc. It contains data properties hasDNASequence. It has instances like tom, ash, bob, etc. The instances of Person can have particular value for its properties like the value of hasDNASequence is AACTGG , the value of hasAge is 19, the value of hasMother is Sally, etc. Person class cannot support property value as a regular expression which provides flexible means for matching strings of text. For example, Person class might have [A]{3}[TGC]{5} as value of hasDNASequence property. OWL

reasoner cannot parse hasDNASequence's value because regular expression describes a set of strings and OWL does not allow set of values for a property. It requires implementing various ways to represent the infinite sets involved in an application. Part of our motivation is to find out an infinite set of information in a real world. We figured out that information in the regular expression for any real world object such as protein sequences, telephone patterns, etc. can be consider as a collection of information. Information on internet is constantly changing and it is too difficult to characterize it in any meaningful way. It can also be a good source of infinite set of information. Thus, we decided to extend OWL to support collections. Collections are a natural part of the world that we need to model such as protein sequences, social security numbers, etc. OWL supports finite set of collections classes. There are two types of collections supported by OWL, unordered collections and ordered collections.

5.1.1.1 Unordered Collections

There are two type of unordered collection: set and bag.

5.2.1.1.1 Set

Set is a collection that cannot contain duplicate elements. It is expressed by linking all the elements to it. In set, multiple identical values of an element will be eliminated.

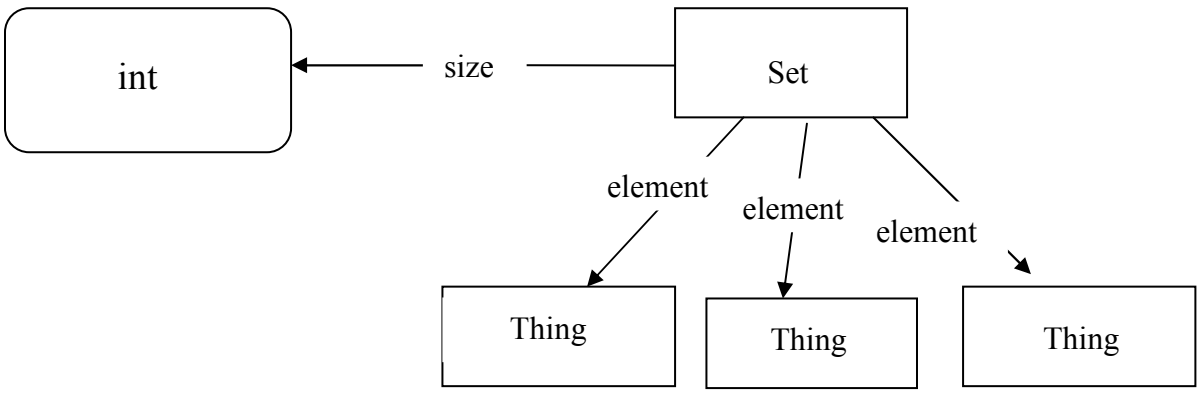

Figure 9: How the Set is implemented

5.1.1.1.2 Bag

Bag is distributed by collections that can have multiple copies of objects. This is performed through the Item entity. The item is linking exactly one resource through the relationship itemResource.

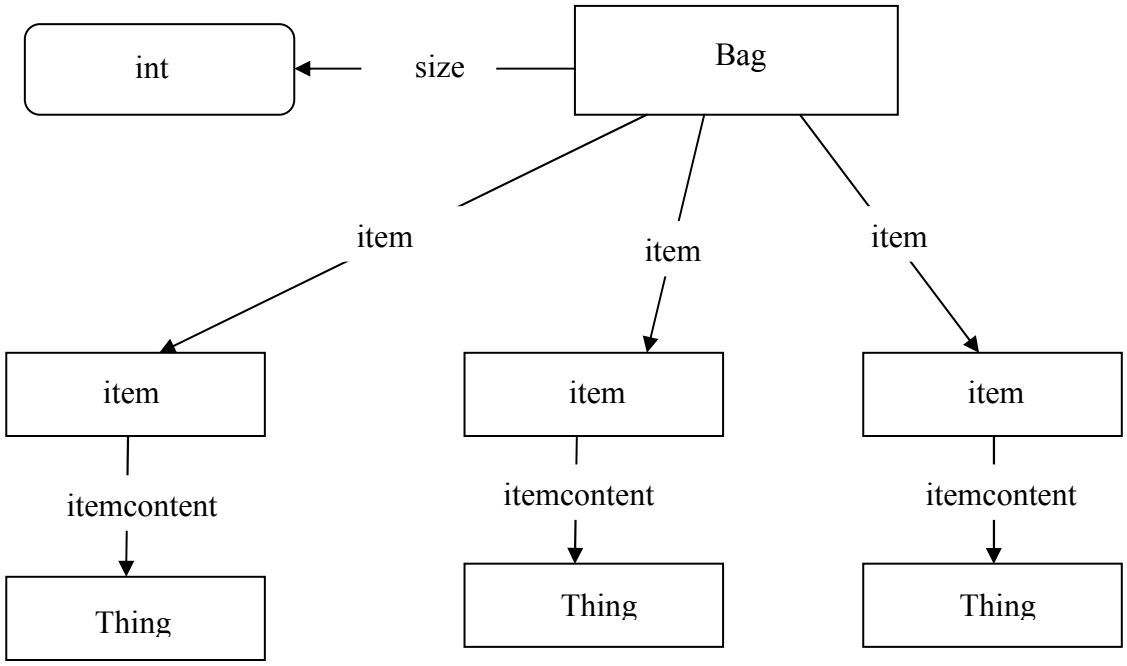

Figure 10: How Bag is implemented

5.1.1.2 Ordered Collections

A List is an ordered collection which tracks the order of the object. It is characterized by collections that can have multiple copies of objects. This is performed through the ListItem. The item is linking exactly one resource through the relationship itemResource.

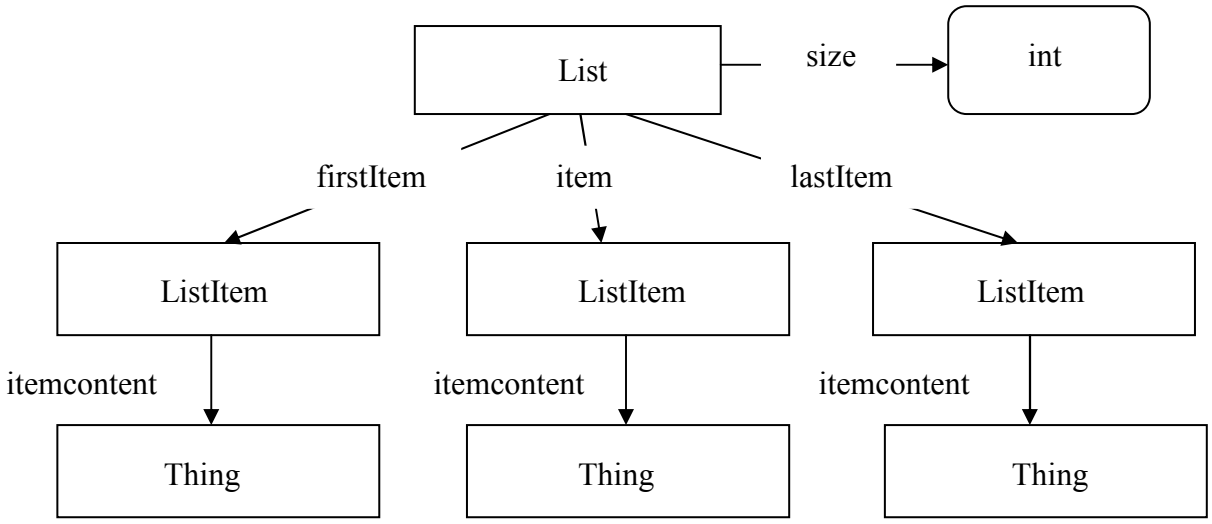

Figure 11: How List is implemented

Consider following denotation in Description Logic:

(subClass (∃hasDNASequence.{ A*TA*G}))
⇒ (Class Person)

The denotation is equivalent to following code in OWL.

```
<owl:Class rdf:about="#Person">
    <rdfs:subClassOf>
            <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDNASequence"/>
                <owl:hasValue rdf:datatype="xsd:string">
                    A*TA*G
                </owl:hasValue>
            </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

In the above example, Person is an OWL class and hasDNASequence is its data property. hasDNASequence has value A*TA*G. It contains regular expression for all of the phone numbers in the USA.

If we added this code to an OWL document and tried to parse the OWL document with Pellet, it would not parse the OWL document and will show errors like unsupported Axiom and

invalid cardinality. The reason behind these errors is that regular expression provides concise and flexible means of matching strings of text. OWL can only support inflexible or fixed string of text or number. Pellet could not parse this code. One reason to use a regular expression was that it can be recognized by finite automata.

In particular, it is difficult to reason directly about infinite sets in OWL. Marek and Remmel showed that the theory of deterministic finite automata (DFA) can be integrated with the theory of set based logic programming to give a setting where one can effectively reason about infinite sets. Suppose that P is a finite set based logic program over a universe Z, where the set represented by atoms in P are languages contained in Z which are accepted by finite automaton and miop O involved in P preserve regular languages, i.e., if A is an automata such that the languages L (A) accepted by A is contained in X, then we can effectively construct an automaton B such that the language L (B) accepted by B equals O (L (P)). Thus, they showed that the stable models of logic program P are languages accepted by finite automaton and one can effectively check whether a language accepted by finite automaton is a stable model. One can effectively reason about infinite sets through this way. Based on this theory, we decided to add finite automata constraints to OWL which involve infinite sets to check whether a language accepted by finite automaton is a stable model [1].

5.1.2 Goal

The goal of this deliverable is to extend OWL to represent information about infinite sets where one can reason about infinite sets. Part of the goal is to add finite automata constraints to OWL. The extension to OWL can support infinite sets and constraints where membership in the collection can be computed by finite automata.

5.1.3 Implementation and Results

One way to extend OWL is to add support for parameterized classes. In this section we discuss such an extension and then give it a semantics based on the Remmel and Marek extension of Answer Set programming with finite automata constraints.

5.1.3.1 Collections of Classes

We have added a new tag CollectionClass to OWL. CollectionClass can contain a regular expression or pattern as its data property value. An OWL Class (i.e. subclass or superclass) is a member of a CollectionClass if all the instances of OWL Class satisfy the CollectionClass's data property. Below is a diagram illustrating one possible use of CollectionClasses.

We have defined specification for CollectionClass in section 3.

The Collection Class Person {A*TA*G}

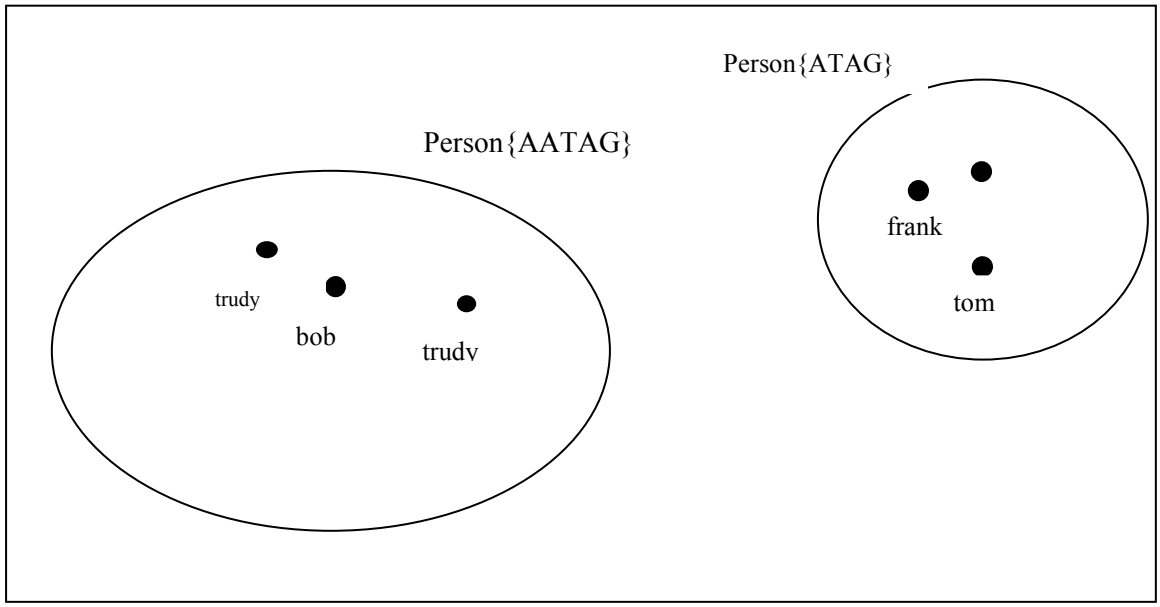

Figure 12: Venn diagram for extended OWL

Figure 12 shows the Venn diagram shows that Person is CollectionClass and it has some property with A*TA*G as its value. This value is regular expression. Person can have a set of subclasses which are having property value which match with property value of CollectionClass

Person. The subclasses of Person has instances which have some fixed value which satisfy CollectionClass's property value.

Consider the following denotation in Description Logic:

(subClass (∃hasDNASequence.{ A*TA*G}))
⇒ (CollectionClass TelephonePattern)

This denotation is equivalent to following description in OWL:

```
<owl:CollectionClass rdf:about="#Person">
   < rdfs:subClassOf>
       <owl:Restriction>
         <owl:onProperty rdf:resource="#hasDNASequence"/>
        <owl:hasValue rdf:datatype="xsd:string">
                A*TA*G</owl:hasValue>
      </owl:Restriction>
     </rdfs:subClassOf>
  </owl:CollectionClass>
```

In the above example, TelephonePattern is a CollectionClass. It has one data property "hasTelephone". It contains regular expression [0-9]{3}-[0-9]{3}-[0-9]{4} as its value.

CollectionClass can have intersection of two other CollectionClasses. Its property value would intersect both of the CollectionClasses property value.

Consider following denotation in Description Logic:

(intersection TelephonePattern SanJoseTelephone)
⇒ (CollectionClass FamilyTelephone)

This denotation is equivalent to following description in OWL:

```
<owl:CollectionClass rdf:about="#FamilyTelephone">
        <owl:equivalentClass>
              <owl:intersectionOf rdf:parseType="Collection">
                    <owl:CollectionClass rdf:about="#TelephonePattern"/>
                     <owl:CollectionClass rdf:about="#SanJoseTelephone"/>
              </owl:intersectionOf>
        </owl:equivalentClass>
 </owl:CollectionClass>
```

In the above example, FamilyTelephone is an intersection of TelephonePattern and SanJoseTelephone. Thus, we have added CollectionClass to OWL for supporting infinite sets

5.1.3.2 To support finite automata constraints

Marek and Remmel[1] showed how one can check whether a language accepted by finite automata is a stable model and one can reason about infinite set in this way. We decided to create a semantics for our CollectionClass's based on these ideas..

We extended OWL's notion of subclass definition. To do this we have added new positive variables as well as constraints in the subclass definition. The syntax for subclass is as follows:

$e \in subClass :- e1 \in c1, e2 \in c2, \ldots, c_1 \in C_1, c_2 \in C_2, \ldots, c_n \in C_n$

$e, e1, e2 \ldots$ are instances. $c1, c2, \ldots$ are OWL classes and $C_1, C2, ...$ are CollectionClasses. We declared entity e by listing its properties. We created ground model for subclass. $e1 \in c1$, $e2 \in c2, \ldots$are positive variables and $c_1 \in C_1, c_2 \in C_2, \ldots$ are constraints. We set the value of every positive variable as true. We checked whether class $c_1, c_2, \ldots, c_n$ are member of CollectionClass $C_1, C_2, \ldots, C_n$ respectively. We guessed true or false for every constraint from 0 to 2^{max} times, where max is number of constraints in subClass. Then we derived whether class $c_1, c_2, \ldots, c_n$ are member of CollectionClass $C_1, C_2, \ldots, C_n$ respectively. We are checking whether property values of all the instance of owlClass can satisfy the same property value of Collection Class. It would give true or false value for every constraint. If the guessed values for constraints are the same as

the derived values then we can conclude that a stable model exists and the model is ready for reasoning. Thus, we have checked membership in CollectionClass by finite automata. We added three axioms to OWL to represent constraints. These axioms are: memberClassOf, collectionClassOf and instanceOf.

5.1.3.2.1 memberClassOf

memberClassOf is used to check whether one class is a member of CollectionClass. It is used with collectionClassOf axiom. Any OWL class can contain one or more memberClassOf axioms. It represents member class which is any OWL class.

For example,

```
<owl:memberClassOf rdf:ID="Daughter">
```

memberClassOf axiom contains Daughter as member class.

5.1.3.2.2 collectionClassOf

collectionClassOf is used with memberClassOf axiom. It represents CollectionClass. memberClassOf and collectionClassOf axioms create one constraint for a logic program.

For example,

```
<owl:memberClassOf rdf:ID="Daughter">
        <owl:collectionClassOf rdf:about="#TelephonePattern"/>
</owl:memberClassOf>
```

Daughter is a member class and TelephonePattern is a CollectionClass. It checks whether Daughter is a member of TelephonePattern.

5.1.3.2.3 instanceOf

instanceOf axiom states that one instance is an object of the class. It is basically used for adding positive variables to the rules of the logic program. It is used with OWL class.

For example,

```
<owl:instanceOf rdf:ID="barackobama">
            <owl:Class rdf:ID="Adult"/>
  </owl:instanceOf>
```

barackobama is an instance and Adult is OWL class. It checks whether grandfather is an object of Senior class.

Let us consider following extended class denotation in DL:

(equiv Son (intersection Male Atmost(2, hasChild))
∧ (member barackobama Adult)
∧ (member michelleobama PersonWithExactlyTwoChilderen)
∧ (member malia Teenager)
∧ (memberClass Daughter TelephonePattern)
∧ (memberClass Person FamilyTelephonePattern)
⇒ (class Son)

Here, barackobama, michelleobama and malia are instances of classes. Adult, PersonWithExactlyTwoChildren, Teenager, Son and Male are Classes. TelephonePattern and FamilyTelephonePattern are CollectionClasses. hasTelephone and hasChild are properties.

The denotation in DL is equivalent to following description of extended class in OWL :

```
<owl:Class rdf:about="#Son">
    <rdfs:label>Son</rdfs:label>
      <rdfs:comment><![CDATA[]]></rdfs:comment>
      <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#Male"/>

                      <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasChild"/>
                        <owl:maxCardinality rdf :datatype=

                     "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
                          2 </owl:maxCardinality>
                      </owl:Restriction>
                  </owl:intersectionOf>
              </owl:Class>
      </owl:equivalentClass>
      <owl:instanceOf rdf:ID="barackobama">
```

```
            <owl:Class rdf:ID="Adult"/>
        </owl:instanceOf>
        <owl:instanceOf rdf:ID="michelleobama">
            <owl:Class rdf:ID="PersonWithExactlyTwoChildren"/>
        </owl:instanceOf>
        <owl:instanceOf rdf:ID="malia">
           <owl:Class rdf:ID="Teanager"/>
        </owl:instanceOf>
         <owl:memberClassOf rdf:ID="Daughter">
             <owl:CollectionClassOf rdf:about="#TelephonePattern"/>
        </owl:memberClassOf>
        <owl:memberClassOf rdf:ID="Person">
                 <owl:CollectionClassOf rdf:about="#FamilyTelephonePattern"/>
        </owl:memberClassOf>
  </owl:Class>
```

In the above example, TelephonePattern and FamilyPattern are CollectionClass which are involved in constraints. Son is an extended subClass. You can see that we have added memberClassOf, collectionClassOf and instanceOf in the definition of Son.

5.1.4 Remarks

The extension to OWL can support collections and constraints that checks membership in collections by finite automata. To create an extension to OWL, we have faced some challenges which are following:

5.1.4.1 Write specification for CollectionClass

As the main goal of this deliverable is to extend OWL to support infinite sets, we had to define some entity to support infinite sets. During our research, we noticed that OWL does not support regular expressions. We decided to add new entity to OWL which can support regular expression. We developed CollectionClass which has regular expression as property value.

5.1.4.2 Write specification for constraints

We wanted to involve collection in constraints. We decided to check the membership of OWL Class in CollectionClass. We had to define constraints which can check membership of OWL Class in CollectionClass. We decided to extend the definition of sub class. We added

axioms like memberOf, collecionClassOf and instanceOf to subClass. Thus, we added constraints where membership in collection is computed by finite automata.

5.2 Deliverable 2: Test extended OWL with inference engine

5.2.1 Motivation

We successfully created an extension to OWL but in order to test our extension, we needed to create inference engine for parsing and reasoning about an OWL document. To test extended OWL document, we developed an inference engine. This inference engine extends Pellet which can parse our newly added entity and axioms to OWL. In addition, this extension to Pellet can parse constraints and compute stable model by checking membership in collections by finite automata.

5.2.2 Goal

The goal of this deliverable is to test an extended OWL document with inference engine. Part of the goal is to develop an inference engine which extends Pellet to support our extension to OWL. This extension to Pellet can compute stable model by checking membership in collection by finite automata and reason about reduced OWL document.

5.2.3 Implementation and Results

We divided this deliverable in three sub tasks: parse an extended OWL document, compute stable model and reason about extended OWL document with inference engine.

5.2.3.1 Parse an extended OWL document

I have developed an extension to OWL but Pellet cannot parse an extended OWL document. To parse an extended OWL document, i extended Pellet. I used DOM API to parse an extended OWL document. I have added one class named `DOMParser` to parse an extended OWL document. I added a method called `parseDocument` which can parse OWL document.

My first task is to parse our extended OWL Class and get constraints from it. I have developed method called `parseClass` which can parse an extended OWL Class. I added `parseMemberClass` method to the `DOMParser` class for parsing constraints and storing them into array. `parseMemberClass` returns array. I also added `parseCollectionClass` to parse CollectionClass. I got all the constraints and stored them in to array. A constraint contains memberClass and CollectionClass.

5.3.2.2 Compute stable model semantics

After parsing the constraints, the most important thing is to compute the stable model semantics. My task is to check membership of memberClass in CollectionClass for computing stable model. If property values of all the instances of memberClass match the same property value of CollectionClass, then we can say that memberClass is a member of CollectionClass. To compute stable model, I implemented method called `checkStableModel`. I passed constraints as an argument to this method. `checkStableModel` finds out CollectionClass and memberClass from the constraints. I added `parseProperty` method to parse property of an element. Element can be CollectionClass, memberClass or instance of memberClass. I also added `getPropertyValue` method to get property value of element. I got property name and value from the CollectionClass by calling parseProperty and getPropertyValue methods in checkStableModel method. Then I parsed all of the instances of memberClass and checked whether property value of CollectionClass matches values of the same property of all the instances of memberClass. If both match then the constraint is satisfied and 1 is returned. `checkStableModel` returns string of 0 or 1.

I guessed for every constraint from 0 to 2^{max}, where max is the total number of constraints in extended OWL class. If these guessing values of constraints match the derived values of constraints then we can say that stable model exists.

5.3.2.3 Reason about extended OWL document

Pellet can reason about a current OWL document but it cannot reason about an extended OWL document. The task is to remove all the new tags from the extended document and then use Pellet to reason about reduced OWL document. To remove extended OWL tags from the document, we implemented `reducedOWL` method that can remove new tags like CollectionClass, memberClassOf, collectionClassOf and instanceOf from the extended OWL document. It writes reduced OWL document. Before getting a reduced OWL document, I checked the existence of a stable model by using checkStableModel method. If it exists then I can write reduced OWL document to new file. Now Pellet can reason about reduced OWL document. I decided to reason about OWL document with three inferences. Three inferences are: Why X is subclass of Y?, Is X subclass of Y? and Is query1 subsumed by query2? First inference is an explanation example, second inference is a logical test example and third inference is a query subsumption example. Thus, I developed an inference engine which extends Pellet to reason about OWL document.

5.2.4 Case Studies

I experimented with various OWL documents to support our extension to OWL. The case studies for supporting our extension to OWL are as follows:

5.2.4.1 Case Study-1

I developed OWL document and ontology for families. It contains OWL classes like `Person, Male, Female, FamilyMembers, Son, Daughter` ,etc. `Person` was a

super class of every other OWL classes. The ontology on families showed relationship between all the family members like Son, Daughter, Father, Mother, etc. Son had instances like bob, sam, and vick. I added one object property called hasTelephone to Person class. hasTelephone could contain string value. bob, sam and vick contained 408-999-9999, 408-222-2222 and 408-333-2222 as the values of hasTelephone respectively. I decided to add the regular expression for telephone number as a value of object property of OWL class because this regular expression could satisfy collection of telephone numbers. I added one new class TelephonePattern to check the membership of other class with it. TelephonePattern was a sub class of Person. It had instances like sanjose and sandiego. Both instances contained regular expression as value for hasTelephone. I used Pellet to reason about OWL document. I tried to parse OWL document on families with Pellet's OWL reasoner but Pellet could not parse it because OWL did not support a property of class which has not any definitive values. As a regular expression satisfies a set of values, it cannot be parsed as a value of object property of OWL class. Then I decided to create new entity which can contain collection of classes as member classes. I added CollectionClass definition to OWL. I added TelephonePattern as CollectionClass to family's ontology. It contained data property hasTelephone which can contain regular expression of telephone number. It was [0-9]{3}-[0-9]{3}-[0-9]{4}. As mentioned before, it has not any instances. Pellet cannot directly reason about TelephonePattern. Therefore, I decided to add constraints to check membership in CollectionClass. I extended a sub class definition. I invented new axioms like memberClassOf, collectionClassOf and instanceOf. I added these axioms in the definition of Son class. memberClassOf contains Son as member Class and collectionClassOf contains TelephonePattern as CollectionClass. To add positive variables, I added instanceOf axiom

to OWL. I extended Pellet to support our extension by using DOM API. I checked the value of `hasTelephone` of Son with the value of `hasTelephone` of `TelephonePattern`. To compute stable model, we guessed values for this constraint. It had two guessed values: false and true. If all values of instances match with the regular expression then it satisfies the constraint. The derived value for this constraint is true and it matches the guessed value when guessed value for the constraint is true. It means that a stable model exists. Then I removed all the extended entities and axioms from extended OWL document and made new reduced OWL document which Pellet can parse. I used three inferences to reason about reduced OWL document with Pellet.

5.2.4.2 Case Study-2

As mentioned in Case Study-1, we developed OWL document for families. I added one new feature to our extension. I added intersection of two or more CollectionClasses in the definition of another CollectionClass. I added two new CollectionClasses : `SanJoseTelephone` and `FamilyPattern`. `SanJoseTelephone` contains 408-[0-9]{3}-[0-9]{3} as a value for hasTelephone. `FamilyPattern` is CollectionClass which contains intersection of `TelephonePattern` and `SanJoseTelephone`. I also added one new constraint which contains `FamilyPattern` as CollectionClass and `Son` as member class. To check membership of `Son` in `FamilyPattern`, I intersected regular expression of `hasTelephone` values of `TelephonePattern` and `SanJoseTelephone` and got new regular expression value. I matched this new regular expression value with every value of `hasTelephone` of `Son`'s instances. I found out that the regular expression matched with all the `hasTelephone` values of instances of Son. I declared two constraints in `Son`. To compute stable model, I guessed values for these constraints for 4 times (0 to 2^2 times). I found that both

constraints were satisfied. Derived values of constraints matched with guessed value of constraints when both the guessed value would be true. I could say that stable model exists for those guessed values. After removing all the extended axioms and entities from OWL document, the document was ready to reason about and Pellet could parse it. Pellet was reasoning with our inferences. Thus, I developed extension which can also accept intersection of two or more CollectionClasses.

5.2.4.3 Case Study-3

To support our extension, I developed OWL document and ontology for computers. This ontology showed the relationship between the parts of computer like `mouse, keyboard, hard disk, memory`, etc. I added OWL Classes like `ComputerType, Mouse, led+mouse, scroll+mouse, optical+mouse, keyboard, memory, harddisk`, etc in computer ontology. As we discussed in Case Study-1, OWL did not support regular expression as a string value. I decided to add regular expression of computer ID number of computer. I added three collection classes: `WindowsPattern, MacPattern and ComputerPattern`. These CollectionClasses contained `hasComputerNumber` as data property. It contains regular expression of computer ID. `WindowsPattern` contained MIS-[0-9]{6} as value of hasComputerNumber. `MacPattern` contained MAC-[0-9]{6} as value of hasComputerNumber. `ComputerPattern` contained ^\w{3}-[0-9]{6} . I extended sub class called `ComputerType`. I added two constraints and three positive variables in the definition of `ComputerType`. First constraint contained `MacPattern` as CollectionClass and `MicrosoftComputer` as member class. Second constraint contained `ComputerPattern` as CollectionClass and `DellComputer` as member class. I guessed values for both the constraints and checked membership of member class in CollectionClass. I found that the first

constraint was not satisfied and the second constraint was satisfied. It meant that derived values of constraints matched with guessed values of constraints when guessed value of first constraint was false and guessed value of second constraint is true. I could say that stable model existed. I removed all the extended tags from the OWL document and created reduced OWL document which Pellet could reason about. I used our three inferences to reason about OWL document by using Pellet's OWL reasoner. Thus, I experimented with computer ontology to support our extension to OWL.

5.2.4.4 Case Study-4

To support our extension to OWL, I created OWL document and ontology for DNA Sequences. This ontology showed that relationship between DNA sequences of species. It contains OWL classes like `Human, Species, Monkey, Rat`, etc. I added two CollectionClasses to this ontology. These CollectionClasses were `HumanPattern` and `AnimalPattern`. These ColletionClasses has data property called `hasDNASequence`. It contained regular expression of DNA. `HumanPattern` contained [A]{3}[C]{2} and `AnimalPattern` contained [G]{4}[T]{2}[C]{1}. I extended sub class `Human`'s definition by adding constraints and positive variables. I added three constraints. First constraint contained `HumanPattern` as CollectionClass and `Human` as memberClass. Second constraint contained `Monkey` as member class and `HumanPattern` as CollectionClass. Third constraint contained `Monkey` as member class and `AnimalPattern` as CollectionClass. I guessed values for these three constraints. I guessed for 8 times (0 to 2^3 times) for computing stable model. I derived values by checking membership of member class in CollectionClass. The first and third constraints were satisfied whereas the second constraint was not satisfied. The stable model was existed when derived values of constraints matched the guessed value of the constraints. The

stable model was existed in case 5 when we guessed true for the first constraint, false for second constraint and true for the third constraint. Then I removed our extended OWL tags from the extended OWL document and created new reduced OWL document. I parsed reduced OWL document with Pellet and used three inferences to reason about with reduced OWL document.

Thus, I could also check membership between DNA sequences with our extension to OWL. This was another example of supporting our extension to OWL.

5.2.5 Outputs

As mentioned before, we have created three inferences. We have developed three examples to check these three inferences. We have reasoned about the family.owl which you can find in Appendix.

5.2.5.1 Explanation Example

I created object of `DOMParser` class in ExplanationExample. I called `parseDocument` method of `DOMParser` to parse whole extended document. This method was computing stable model. If stable model existed then it removed our extended stuff from the extended OWL document and creating new reduced OWL document. In explanation example, I created OWL API manager that allowed loading an ontology file and created OWL entities. I created a renderer to print the explanation and reasoner to reason. I loaded the ontology by using reasoner. To generate explanation, I called explanation generator. To show meaningful knowledge of OWL document, I created concepts. I derived entities like OWL class from ontology. I derived two classes: `Son` and `Child+Male`. Explanation generator generates explanation for these two classes. Our inference was "Why Son is sub class of Child + Male?". Pellet's explanation generator called its `getSubClassExplanations` method. This method returns sets of axioms and parsed domain and range for axioms. It also parsed value of axioms

and returned equivalent class from the axioms of the given class. It compared axioms values and returned explanation if it found matches. Figure 13 shows the sub class explanation for `Son` and `Child+Male`.

```
<terminated> ExplanationExample (3) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe
OWL Document is ready for answering
Output File has been written
Why is Son subclass of Child+Male?
Explanations (2):
1)    Male subClassOf Child+Male
      Son equivalentTo Male
                        and hasChild min 2

2)    Child+Male equivalentTo Male
                              and hasChild some Father+Son
      hasChild range Father+Son
      Son equivalentTo Male
                        and hasChild min 2
```

Figure 13: Pellet Explanation Example

5.2.5.2 Query Subsumption Example

Query subsumption example is used to work with queries of OWL document. I called `parseDocument` method of `DOMParser` to compute stable model and create reduced OWL document. I created ontology model to read and prepare ontology for reasoning. I have initialized graph for knowledge from ontology model. I have also initialized parser from Pellet's `Query Engine`. I have also created two queries by using Pellet's `Query` Class. To check query subsumption, I have used `isSubsumedBy` method of `Query Engine`. I passed both of the queries as parameters to `isSubsumedBy`. It returns true if first query is subsumed by second query. I have used family.owl with this inference. Figure 14 shows the output of query subsumption example. In first query, we checked Male of family ontology. I checked Person of family ontology in second query. Pellet derived that say that every `Male` would be `Person` but

every `Person` would not be `Male`. So Query 1 was subsumed by Query 2 but Query 2 was not subsumed by Query 1.

```
<terminated> QuerySubsumptionExample (1) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe
OWL Document is ready for answering
Output File has been written
Example 1
=========
Query 1: query(?x) :- family:Male(?x).
Query 2: query(?x) :- family:Person(?x).

Query 1 is subsumed by query 2: true
Query 2 is subsumed by query 1: false
```

Figure 14: Pellet Query Subsumption Example

5.2.5.3 Logical Example

Logical Example is the last example of reasoning about OWL document. As mentioned before, I first created ontology model to read and prepare ontology. I then parsed OWL class by its name. I derived property name and its value of that class. I compared values of same property of classes. Figure 15 shows the logical example. I have reasoned about family.owl. I parsed `PersonWithAtLeastTwoFemaleChildren` and `PersionWithAtLeastTwoChildren`. I got `hasTelephone` property value of both the classes. I compared both the values. If it matches then it returns true otherwise it returns false. I had implemented `checkSubClass` method. It returned Boolean value. `checkSubClass` checked property value of class with the property value of other class. I got list of subclasses of given OWL class by using `listSubClasses` of OWL Class. listSubClasses listed all the sub class of the OWL class.

```
<terminated> MembershipTest (4) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe (Nov 20, 2010 7:11:49 PM)
OWL Document is ready for answering
Output File has been written

Is PersonWithAtLeastTwoFemaleChildren subClass of PersonWithAtLeastTwoChildren?
Yes, PersonWithAtLeastTwoFemaleChildren is subClass of PersonWithAtLeastTwoChildren
```

Figure 14: Pellet Logical Example

5.2.6 Remarks

Working with this deliverable gave me a very clear understanding of the working of Pellet and DOM API. We extended Pellet by adding `DOMParser` Class to it. Pellet is an efficient OWL reasoner and it is easy to use. I recommend its use for reasoning OWL document as Pellet is open source. I computed a stable model by checking membership in collection by finite automata. I have faced following challenges to test extended OWL document with Pellet:

5.2.6.1 Parse extended OWL document

Pellet could not support our extension to OWL. I tried to parse extended OWL document by making some changes in Pellet but it was hard to build Pellet. First, I tried with Pellet's older version Pellet nightly. I extended Pellet nightly and it was built by using ANT but it is not supporting all the inferences. I decided to use Pellet's latest version. While extending OWL, I also decided to extend Pellet. I added `DOMParser` class to Pellet. After computing stable model semantics, I have removed all the extended tags from OWL document. Pellet can easily reason about that reduced OWL document.

5.2.6.2 Add intersection of CollectionClass

While extending OWL to support collection, I decided to add intersection of two CollectionClasses but I required parsing it as well as finding intersection of both the CollectionClasses. I used DOM API to parse intersectionOf axiom. I made changes in

checkStable function to find intersection of two or more CollectionClasses. I computed new regular expression by intersecting the regular expression of two or more CollectionClasses. The new regular expression would be property value of CollectionClass.

5.2.6.3 Selection of Inferences

I have extended OWL to support collection and constraints. To reason about extended OWL document, we decided to create inference. The goal was to show the meaningful representation of knowledge of ontology. Pellet had supported two inferences: Explanation and Query subsumption. I thought to go with both of these inferences. I also added one more inference to reason about extended OWL document. It was logical inference. Explanation inference explained the reason why one class was the sub class of other class. Query subsumption checked whether query 1 subsumed by query2 and vice versa. Logical inference checked whether one class was sub class of other class.

6 Conclusion

We considered using answer set programming and OWL as they both are used for knowledge representation. One recent extension to answer set programming is to support representing information about infinite sets and constraints where membership in the sets can be computed by finite automata. In this extension, new types of constraints are introduced that allow for a more compact representation of problems in answer set programming. We experimented and observed that OWL did not support infinite set of collections. We attempted to extend OWL to support infinite sets and finite automata constraints. We created a new entity called CollectionClass to support collections. The extension to OWL for allowing infinite sets lead to undecidable reasoning procedures. We created new axioms called memberClassOf, instanceOf and collectionClassOf. In our extension to OWL, we added constraints which involve collections to compute stable model semantics. We extended definition of OWL sub class that can support constraints where membership in the collections could be computed by finite automata. We computed stable model semantics by checking membership in the collections by finite automata. To support our extension to OWL, we extended Pellet to reason about extended OWL document. We have written four test OWL documents to support our extension to OWL. In case study-1 and 3, we extended subClass definition by adding positive variables and constraints. In case study-2, we added intersection of two or more CollectionClasses in CollectionClass definition. In case study-4, we experimented on ontology for DNA sequences.

7 References

[1] Victor Marek and Jeffery B. Remmel, Automata and Answer Set Programming,2009.

[2] L.Niemela, P.Simons and T. Syrjanen, Smodels: A System for Answer Set Programming,2000.

[3] D.Cenzer, J.Remmel and V.Marek, Logic programming with infinite sets,2005.

[4] A.Rector, R.Stevent and G.Moulton, Putting OWL in order: Pattern for sequences in OWL,2003.

[5] Y.Ding, D.Embley and S.Liddle, OWL-AA: Enriching OWL with instance recognition semantics for automated semantic annotation,2005.

[6] M. Gelfond and V. Lifschitz, The Stable Model Semantics for logic programming. In Proceedings of the Fifth Logic Programming Symposium, pages 1070-1080. The MIT Press, 1988.

[7] W.Chen and D. Warren, Computation of Stable Models and its integration with logical Query Procession.

[8] V. Marek and M.Trunszczynsk, Stable Model and an alternative logic programming paradigm.

[9] N.Drummond, A.Rector, R.Stevens and G.Moulton, Putting OWL in order: Patterns for sequences in OWL,2006.

[10] B. Motik, P. Patel-Schneider and I. Harrocks (2007). OWL 1.1 Web Ontology Language Structural Specification and Functional-Style Syntax [Online].Available: http://www.webont.org/owl/1.1/owl_specification.html

[11] D. McGuinness and F. Harmelen (2004). OWL Web Ontology Language Overview [Online]. Available: http://www.w3.org/TR/owl-features/

[12] Pellet OWL Reasoner [Online]. Available:

http://www.mindswap.org/2003/pellet/index.shtml

Appendix

family.owl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xmlns:ns0="http://cohse.semanticweb.org/ontologies/family#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xml:base="http://cohse.semanticweb.org/ontologies/family"
   xmlns="http://cohse.semanticweb.org/ontologies/family#">
   <owl:Ontology rdf:about=""/>
   <owl:CollectionClass rdf:about="#TelephonePattern">
     <rdfs:subClassOf>
           <owl:Restriction>
             <owl:onProperty rdf:resource="#hasTelephone"/>
             <owl:hasValue rdf:datatype="xsd:string">[0-9]{3}-[0-9]{3}-[0-9]{4}</owl:hasValue>
           </owl:Restriction>
      </rdfs:subClassOf>
   </owl:CollectionClass>
   <owl:CollectionClass rdf:about="#WashingtonDCTelephone">
        <rdfs:subClassOf>
              <owl:Restriction>
                <owl:onProperty rdf:resource="#hasTelephone"/>
                <owl:hasValue rdf:datatype="xsd:string">202-[0-9]{3}-[0-9]{2}</owl:hasValue>
              </owl:Restriction>
         </rdfs:subClassOf>
   </owl:CollectionClass>
   <owl:CollectionClass rdf:about="#FamilyTelephone">
        <owl:equivalentClass>
              <owl:intersectionOf rdf:parseType="Collection">
                   <owl:CollectionClass rdf:about="#TelephonePattern"/>
                   <owl:CollectionClass rdf:about="#WashingtonDCTelephone"/>
              </owl:intersectionOf>
        </owl:equivalentClass>
   </owl:CollectionClass>
   <owl:Class rdf:about="#Person">
          <rdfs:label>Person</rdfs:label>
           <rdfs:comment><![CDATA[]]></rdfs:comment>
           <owl:equivalentClass>
              <owl:Class>
                 <owl:disjointUnionOf rdf:parseType="Collection">
                    <owl:Class rdf:ID="Male"/>
                         <owl:Class rdf:ID="Female"/>
```

```
                    <owl:Restriction>
                       <owl:onProperty rdf:resource="#hasFather"/>
                       <owl:someValuesFrom>
                          <owl:Class rdf:about="#Male"/>
                       </owl:someValuesFrom>
                    </owl:Restriction>
                 </owl:disjointUnionOf>
              </owl:Class>
            </owl:equivalentClass>
            <rdfs:subClassOf>
               <owl:Restriction>
                  <owl:onProperty rdf:resource="#hasTelephone"/>
                  <owl:cardinality rdf:datatype="xsd:string">1</owl:cardinality>
               </owl:Restriction>
            </rdfs:subClassOf>
   </owl:Class>
  <owl:Class rdf:about="#Son">
        <rdfs:label>Son</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <owl:equivalentClass>
                <owl:Class>
                   <owl:intersectionOf rdf:parseType="Collection">
                      <owl:Class rdf:about="#Male"/>
                      <owl:Restriction>
                         <owl:onProperty rdf:resource="#hasChild"/>
                         <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">2</owl:minCardinality>
                      </owl:Restriction>
                   </owl:intersectionOf>
                </owl:Class>
        </owl:equivalentClass>
        <owl:memberClassOf rdf:ID="Person">
                <owl:collectionClassOf rdf:about="#TelephonePattern"/>
        </owl:memberClassOf>
        <owl:memberClassOf rdf:ID="Daughter">
                 <owl:collectionClassOf rdf:about="#FamilyTelephone"/>
        </owl:memberClassOf>
        <owl:instanceOf rdf:ID="michelleobama">
            <owl:Class rdf:ID="Female"/>
        </owl:instanceOf>
        <owl:instanceOf rdf:ID="barackobama">
           <owl:Class rdf:ID="PersonWithExactlyTwoChildren"/>
        </owl:instanceOf>
        <owl:instanceOf rdf:ID="malia">
           <owl:Class rdf:ID="Teanager"/>
        </owl:instanceOf>
```

```
  </owl:Class>
  <owl:Class rdf:about="#Daughter">
            <rdfs:label>Daughter</rdfs:label>
        <rdfs:comment><![CDATA[]]></rdfs:comment>
        <owl:equivalentClass>
                  <owl:Class>
                     <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Female"/>
                          <owl:Restriction>
                           <owl:onProperty rdf:resource="#hasChild"/>
                           <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">2</owl:minCardinality>
                          </owl:Restriction>
                      </owl:intersectionOf>
                  </owl:Class>
          </owl:equivalentClass>
   </owl:Class>
    <owl:Class rdf:about="#PersonWithExactlyTwoChildren">

        <owl:equivalentClass>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:cardinality rdf:datatype="xsd:nonNegativeInteger">2</owl:cardinality>
          </owl:Restriction>
        </owl:equivalentClass>
    </owl:Class>
    <owl:Class rdf:about="#Child+Male">
          <rdfs:label>Male</rdfs:label>
            <rdfs:comment><![CDATA[]]></rdfs:comment>
            <owl:equivalentClass>
               <owl:Class>
                  <owl:intersectionOf rdf:parseType="Collection">
                     <owl:Class rdf:about="#Male"/>
                     <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasChild"/>
                        <owl:someValuesFrom>
                           <owl:Class rdf:about="#Father+Son"/>
                        </owl:someValuesFrom>
                     </owl:Restriction>
                  </owl:intersectionOf>
               </owl:Class>
              </owl:equivalentClass>
             <owl:Restriction>
                        <owl:onProperty rdf:resource="#isMarriedTo"/>
                        <owl:someValuesFrom>
                           <owl:Class rdf:about="#Female"/>
```

```
                    </owl:someValuesFrom>
            </owl:Restriction>
  </owl:Class>
  <owl:Class rdf:about="#Female">
        <rdfs:label>Male</rdfs:label>
   <rdfs:comment><![CDATA[]]></rdfs:comment>
      <owl:Restriction>
              <owl:onProperty rdf:resource="#isMarriedTo"/>
              <owl:someValuesFrom>
                    <owl:Class rdf:about="#Female"/>
              </owl:someValuesFrom>
             </owl:Restriction>
  </owl:Class>
 <owl:Class rdf:about="#Father+Son">
           <rdfs:label>fatherofSon</rdfs:label>
           <rdfs:comment><![CDATA[]]></rdfs:comment>
           <rdfs:subClassOf>
              <owl:Restriction>
                 <owl:onProperty rdf:resource="#uses"/>
                 <owl:someValuesFrom>
                    <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
                 </owl:someValuesFrom>
              </owl:Restriction>
           </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#adult">
     <rdfs:label>adult</rdfs:label>
     <rdfs:comment><![CDATA[Things that are adult.]]></rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="#brain">
     <rdfs:label>brain</rdfs:label>
     <rdfs:comment><![CDATA[]]></rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="#young">
     <rdfs:label>young</rdfs:label>
     <rdfs:comment><![CDATA[]]></rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="#Male">
           <rdfs:label>computer</rdfs:label>
           <rdfs:comment><![CDATA[]]></rdfs:comment>
           <rdfs:subClassOf>
              <owl:Class rdf:about="#Child+Male"/>
           </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:about="#hasChild">
        <rdfs:comment><![CDATA[Anyone that has a pet must like that pet. ]]></rdfs:comment>
```

```
  <rdfs:label>hasChild</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#hasFather"/>
  <rdfs:domain>
        <owl:Class rdf:about="#Person"/>
  </rdfs:domain>
  <rdfs:range>
           <owl:Class rdf:about="#Father+Son"/>
       </rdfs:range>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#uses">
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:label>uses</rdfs:label>
    <owl:inverseOf rdf:resource="#used_by"/>
    <rdfs:domain>
       <owl:Class rdf:about="#optical"/>
    </rdfs:domain>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasFather">
      <rdfs:comment><![CDATA[]]></rdfs:comment>
<rdfs:label>hasFather</rdfs:label>
<rdfs:domain>
       <owl:Class rdf:about="#Male"/>
      </rdfs:domain>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasMother">
         <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:label>hasMother</rdfs:label>
    <rdfs:domain>
          <owl:Class rdf:about="#Female"/>
         </rdfs:domain>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#isMarriedTo">
      <rdfs:comment><![CDATA[]]></rdfs:comment>
<rdfs:label>isMarriedTo</rdfs:label>
      <rdfs:domain>
           <owl:Class rdf:about="#Person"/>
      </rdfs:domain>
</owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#works_for">
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:label>works_for</rdfs:label>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#has_parent">
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:label>has_parent</rdfs:label>
 </owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:about="#likes">
     <rdfs:comment><![CDATA[]]></rdfs:comment>
     <rdfs:label>likes</rdfs:label>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#Child_of">
          <rdfs:comment><![CDATA[]]></rdfs:comment>
          <rdfs:label>Age_of</rdfs:label>
     </owl:ObjectProperty>
     <owl:ObjectProperty rdf:about="#hasChild">
        <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:label>hasAge</rdfs:label>
    <owl:inverseOf rdf:resource="#Child_of"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#is_pet_of">
     <rdfs:comment><![CDATA[]]></rdfs:comment>
     <rdfs:label>is_pet_of</rdfs:label>
     <owl:inverseOf rdf:resource="#has_pet"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="#hasTelephone">
        <rdfs:range rdf:resource="xsd;string"/>
  </owl:DatatypeProperty>
  <family:Person rdf:about="#michelleobama">
       <family:hasAge rdf:datatype="xsd:int">45</family:hasAge>
       <family:hasDaughter rdf:resource="#"/>
       <family:hasTelephone rdf:datatype="xsd:string">202-111-2222</family:hasTelephone>
  </family:Person>
 <family:Person rdf:about="#barackobama">
      <family:hasAge rdf:datatype="xsd:int">49</family:hasAge>
      <family:hasTelephone rdf:datatype="xsd:string">202-111-2233</family:hasTelephone>
 </family:Person>
 <family:Daughter rdf:about="#maliaobama">
      <family:hasAge rdf:datatype="xsd:int">18</family:hasAge>
      <family:hasTelephone rdf:datatype="xsd:string">202-750-4967</family:hasTelephone>
 </family:Daughter>
 <family:Male rdf:about="#bob">
      <family:hasAge rdf:datatype="xsd:int">17</family:hasAge>
      <family:hasTelephone rdf:datatype="xsd:string">408-750-4967</family:hasTelephone>
 </family:Male>
 <family:Female rdf:about="#alice">
       <family:hasAge rdf:datatype="xsd:int">17</family:hasAge>
       <family:hasTelephone rdf:datatype="xsd:string">408-222-1456</family:hasTelephone>
 </family:Female>
</rdf:RDF>
```