Research Article
A Novel Way to Relate Ontology Classes

Ami T. Choksi and Devesh C. Jinwala

Computer Engineering Department, Sardar Vallabhbhai National Institute of Technology, Surat, Gujarat 395007, India

Correspondence should be addressed to Ami T. Choksi; amichoksi.phd@gmail.com

Received 14 December 2014; Accepted 12 March 2015

Academic Editor: Rafael Valencia-García

Copyright © 2015 A. T. Choksi and D. C. Jinwala. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The existing ontologies in the semantic web typically have anonymous union and intersection classes. The anonymous classes are limited in scope and may not be part of the whole inference process. The tools, namely, the pellet, the jena, and the protégé, interpret collection classes as (a) equivalent/subclasses of union class and (b) superclasses of intersection class. As a result, there is a possibility that the tools will produce error prone inference results for relations, namely, sub-, union, intersection, equivalent relations, and those dependent on these relations, namely, complement. To verify whether a class is complement of other involves utilization of sub- and equivalent relations. Motivated by the same, we (i) refine the test data set of the conference ontology by adding named, union, and intersection classes and (ii) propose a match algorithm to (a) calculate corrected subclasses list, (b) correctly relate intersection and union classes with their collection classes, and (c) match union, intersection, sub-, complement, and equivalent classes in a proper sequence, to avoid error prone match results. We compare the results of our algorithms with those of a candidate reasoner, namely, the pellet reasoner. To the best of our knowledge, ours is a unique attempt in establishing a novel way to relate ontology classes.

1. Introduction

The semantic web [1–4] enables computers to understand the meaning or semantics of the information available on the web. The semantics have greatly benefited the research of integration, search, and analysis of data. The semantic web is a vision of “web of linked data.” The Linked data are represented by structured or semistructured ontologies and are described by languages, namely, Resource Description Framework (RDF) [5] and web ontology language (OWL) [6–8]. An ontology is a formal specification of the conceptualization. The concept is represented using classes and properties and relations among them. The classes are of different types, namely, subclass, superclass, intersection class, union class, and complement class. The properties are mainly of two types, namely, data property and object property.

The vendors publish resource ontologies in the semantic web repositories. The user, who wants to discover ontologies, specifies the description of the required one. The resources in repositories are called advertised resources and user’s required resource description is named as requested resource.

The semantic description of resources is preferred over syntactic description, because of being more expressive, sharable, and maintainable [9–11].

The resource discovery is a process of finding the advertised resource(s), those which are relevant to the request [12–15]. The unlimited size of the semantic web makes the resource discovery a challenging task in the semantic web [16].

The existing ontologies in literature typically use anonymous union and intersection classes. The anonymous classes are limited in scope. They may not be part of the whole inference process by ontology reasoners [17]. Therefore, there is a need to use named union and intersection classes in the ontology.

As we observe, the inference based on the union and the intersection classes are partially available in the tools, namely, the jena semantic framework [18, 19], the Pellet reasoner [20], and the protégé editor [21, 22]. The collection classes are interpreted as (i) either subclasses or superclasses of union and intersection classes, respectively, by the tools, namely, the jena and the Pellet and (ii) equivalent classes of union
class by the ontology editor, namely, the protégé. Both of the interpretations are conceptually error prone, as (a) if a class is going to be interpreted as the equivalent/superclass, why should we make it the union class? and (b) if a class is going to be interpreted as a subclass, why should we make it the intersection class? In the example subsection we discuss how the above interpretations are conceptually error prone using an illustration. We believe, unless we specifically mention to treat the class as equivalent, subclass, or superclass, they should be interpreted in their original form ONLY. Therefore, a need arises to establish a proper relation between union and intersection classes and their collection classes. This indeed will affect the sub- and equivalent classes list of any class in ontology, wherein union and intersection classes are part of the ontology, which we believe. This too will affect the complement class list as it is dependent on sub- and equivalent classes internally. Therefore, there is a need to calculate correct sub-, equivalent, and complement classes list of all the classes in the ontology. We believe it can be achieved through proper sequence of matching above classes in a match algorithm.

We, therefore, attempt to refine our test application data set of the conference management system ontology [23] by adding the named, union, and intersection classes. We propose a match algorithm to (a) relate collection classes to their union and intersection classes in their original form only, (b) refine the subclasses list of any class of the ontology wherein named union and intersection classes are part of the ontology tree, to rectify inferring process, and (c) match complement, union, intersection, and equivalent classes in a proper sequence to avoid error prone match results.

We compare the results of our algorithm with the candidate reasoner, namely, the Pellet. To the best of our knowledge, ours is a unique attempt in rectifying the inferring process for named, union, and intersection classes.

Our contributions in the paper are as follows.

(i) We refine the conference ontology [23] of our test application of the conference management system by adding named, intersection, and union classes.

(ii) We propose an algorithm to

(a) correctly relate named, intersection, and union classes to their collection classes,

(b) refine the subclasses list of any class of the ontology wherein named union and intersection classes are part of the ontology tree, to rectify inferring process,

(c) match ontology classes in a proper sequence for sub-, complement, equivalent, union, and intersection classes.

(iii) We compare the results of our algorithm with those of the candidate reasoner, namely, the Pellet.

The organization of the paper includes 7 sections. The first one we have already described, that is, the Introduction. The subsection of Introduction is the example, wherein we discuss the error prone interpretations of tools, namely, the protégé, the jena and the Pellet. In Section 2, we present the theoretical background, in Section 3 we describe the problem statement, in Section 4, we present our proposed algorithm, in Section 5, we describe methodology of implementation, in Section 6, we describe performance results and analysis, and in Section 7, we present the conclusion and future work.

1.1. The Example. An illustration shows the error prone results achieved in interpreting union class as superclass and intersection class as subclass of collection classes.

The Student and Employee are two classes with properties and individuals as follows:

(1) Student (class) is as follows:

(i) Properties = {name, rollno, branch}.

(ii) Individuals = {Ami, Abhay, Aayushi, Ashish}.

(2) Employee (class) is as follows:

(i) Properties = {name, EmployeeID, department}.

(ii) Individuals = {Ami, Ashish, Tushar}.

Let us take the intersection of the classes. That is, all the individuals and properties which are common in both Student and Employee. Let us name intersection class as StudentNEmployee. The properties and individuals of the StudentNEmployee class should be as follows.

(3) StudentNEmployee (intersection class) is as follows:

(i) Properties = {name}.

(ii) Individuals = {Ami, Ashish}.

Let us take the union of the classes now. That is, all the individuals and properties which are either Student or Employee as well as those who are both Student and Employee. Let us name union class as StudentOrEmployee. The properties and individuals of the StudentOrEmployee class should be as follows.

(4) StudentOrEmployee (union class) is as follows:

(i) Properties = {name, rollno, EmployeeID, branch, department}.

(ii) Individuals = {Ami, Abhay, Aayushi, Ashish, Tushar}.

Let us consider MechanicalStudent as the subclass of Student class. The properties and individuals of the class are as follows.

(5) MechanicalStudent (subclass) is as follows:

(i) Properties = {name, rollno, branch, apprnoID}.

(ii) Individuals = {Ashish}.

The properties in bold are inherited from Student class, and they too are part of MechanicalStudent.

From the definition of subclass [24], one can say subclass has more properties/characteristics than its base class. From the definition of equivalent class, one can say the classes should have equal individuals. If we interpret
(i) **collection classes as sub classes of union class**, the `Student` class should have more properties than the `StudentOrEmployee` class. According to the 1st and 4th points, we can say, the inferred fact is error prone;

(ii) **intersection class as the sub class of the collection classes**, the `StudentNEmployee` class should have more properties than `Student` class. From the 1st and 3rd points, we can say, the inferred fact is error prone;

(iii) **collection classes as equivalent classes of union class**, the `Student` class should have same individuals as the `StudentOrEmployee` class. From the 1st and 4th points, we can say, the inferred fact is error prone.

The error prone interpretations of union and intersection classes as super classes or equivalent classes/subclasses of collection classes, in turn, will create error prone sub- and equivalent classes lists. The match process, wherein the sub- and/or equivalent classes are involved, will also produce error prone match results.

### 2. Theoretical Background and Literature Survey

The resource discovery in the semantic web consists of ontology resources. The resource ontology is described using individuals, classes, attributes/properties, relations, restrictions, rules, and axioms like components.

Few terminologies for understanding matching algorithms are defined as follows.

#### 2.1. Terminologies

The following terms are useful in understanding the paper.

1. **Subclass:** A class C1 is defined as a subclass of class C2, which means the set of individuals of C1 should be a subset of the individuals of C2.

2. **Superclass:** A class C1 is defined as a superclass of class C2, which means the set of individuals of C2 should be a subset of the individuals of C1.

3. **Complement class:** A class C1 is defined as complement of class C2, if C1 contains exactly those individuals that do not belong to C2. ComplementOf in OWL is the logical negation. It is the same as if the NOT operator of set theory [25] is applied to classes.

4. **Equivalent class:** A class is said to be equivalent class if they have the same individuals.

5. **Union class:** A union class is having all individuals of its collection classes at least once. It is the same as if the OR operator of set theory applied to classes. The collection classes are said to be operands too.

6. **Intersection class:** An intersection class is having only those individuals which are common in all of its collection classes. It is the same as if the AND operator of set theory is applied to classes.

7. **Equivalent class:** If class A and class B are equivalent classes. They have exactly the same set of individuals.

(viii) **Anonymous class:** An anonymous class is a class without a name/identifier.

(ix) **Resource discovery:** Resource discovery is a process of finding relevant resource for the application under consideration.

(x) **Precision ratio:** The ratio of the number of relevant retrieved resources to the total number of resources retrieved is a precision ratio [26]. Consider

\[
\text{Precision Ratio} = \frac{|\text{relevant resources} \cap \text{retrieved resources}|}{|\text{retrieved resources}|}.
\]

(xi) **Recall ratio:** The ratio of the number of relevant retrieved resources to the number of relevant resources is a recall ratio [26]. Consider

\[
\text{Recall Ratio} = \frac{|\text{relevant resources} \cap \text{retrieved resources}|}{|\text{relevant resources}|}.
\]

(xii) **Matching:** Matching is the process of finding correspondences between semantically related entities of different resources. Advertised resource is considered to be A and requested resource is considered to be R. Properties are considered to have P1, P2, ..., Pn for A and P1, P2, ..., Pm for R. Matching [27–29] can be exact, subsume, plug-in, or no-match. The terminologies used in this work are as follows.

(a) **Exact match:** if all the properties of the advertised resource match with those of the requested one, it is called exact match. Consider

\[
A (\text{PROPERTIES}) = R (\text{PROPERTIES}) \quad \longrightarrow \{ A (P_i) \equiv R (P_i) \}.
\]

(b) **Subsume match:** if advertised resource properties are less than those of requested resource, it is called subsume match. Consider

\[
A (\text{PROPERTIES}) \subseteq R (\text{PROPERTIES}) \quad \longrightarrow \{ A (P_i) \subseteq R (P_i) \}.
\]

(c) **Plug-in match:** if advertised resource properties are greater than those of the requested resource, it is called plug-in match. Consider

\[
A (\text{PROPERTIES}) \supseteq R (\text{PROPERTIES}) \quad \longrightarrow \{ A (P_i) \supseteq R (P_i) \}.
\]

(d) **No match:** if advertised resource properties and requested resource properties do not have any of the above relations, it is called No match. Finding match can be done using semantic similarity [30–33], semantic distance [34, 35], and feature matching.
(xiii) Semantic similarity: a similarity function is defined as a real valued function as follows:

\[
sim(x, y) : \mathbb{C} \times \mathbb{C} \rightarrow [0, 1].
\] (6)

On a set measuring the degree of similarity between \(x\) and \(y\), where \(C\) is the set of well-formed concepts and where \(\sim(x, y)\) measure the degree to which \(y\) is similar to \(x\), \(\sim(x, y) = 1\) means fully similar, namely, the case \(x = y; \sim(x, y) = 0\) means nothing between two concepts. Consider

\[
\text{Sim}(x, y) = p \left( \frac{|A(x) \cap B(x)|}{A(x)} \right) + (1 - p) \left( \frac{|A(x) \cap B(x)|}{A(y)} \right).
\] (7)

(xiv) Structural matching: structural matching [36] involves matching of parameters, namely, ancestors, descendants, leaves, addcates, ascopath, dispath, siblings or subclasses list, and similarity contribution.

(xv) Ontology alignment: it is a process of establishing a collection of binary relation between the vocabularies of two ontologies. Ontology alignment is applied to the same or related domain ontologies. Ontology alignment [37–39] techniques are broadly categorized into terminological, structural, extensional, and semantic techniques.

(xvi) Reasoner: a reasoner or inference engine is used to derive the facts which are true but syntactically difficult to achieve. A reasoner can improve efficiency of matching.

2.2. Related Work. A resource discovery is a matching process of requested and advertised resources. Resources in the semantic web are ontologies. The classes and properties of two ontologies are matched using linguistic, string distances, structural, or combined techniques [37–40].

We observe, in the ontologies available in the literature, the union and intersection classes are typically used as anonymous classes. The scope of anonymous classes is limited; they may not participate in whole inference process of a reasoner [17]. Therefore, there is a need for named union and intersection classes.

The reasoners, namely, Pellet, Fact++, RacerPro, Hermit, and so forth [41] are used to infer the hidden information from the ontologies. The Pellet is a better reasoner from points of view of the reasoning characteristics, practical usability, and performance. The comparison of reasoners is discussed in [20, 41].

Similar to the reasoner, the jena semantic framework too has query engine in it. Therefore, we are motivated to check the inference support of tools, namely, jena and Pellet for the named union and intersection classes. The tools have partial support for the named union and intersection classes [19]. The tools support inference for individuals of union and intersection classes. Let us compare query engines of the tools, namely, Pellet and jena [42], as shown in Table 1. The table shows query engine of Pellet is better than that of jena.

Though the Pellet's query engine is better than that of jena, yet they both interpret collection classes of union and intersection as subclasses and superclasses, respectively.

The designers create ontologies using ontology editors. The characteristics of support for Java, open source, and support for creating simple as well as complex ontologies make the prot´eg´e editor performs better than other existing editors, namely, Apollo, OntoStudio, Swoop, and TopBraid. The comparison of ontology editors is discussed in [21, 22]. We need to check whether named union and intersection classes are added properly or not.

From above points of discussion, we use Pellet reasoner and prot´eg´e editor as our candidate tools for our work. The comparison in Table 2 shows how the tools, namely, prot´eg´e editor and Pellet reasoner interpret the union and the intersection classes.

As discussed in The Example subsection, such interpretation may lead to inference of error prone subclasses and equivalent classes list. In turn, the error prone list will affect other operations which use the sub and equivalent classes list. For example, complement class matching in turn uses sub classes as explained in [43], and ontology structure matching algorithms too uses sub classes list as one of the matching features [36].

Therefore, there is a need to have match algorithm to establish correct interpretations of named union and intersection classes, as well as proper relation with their collection classes. To avoid error prone match results of the algorithm involving sub-, union, intersection, complement, and equivalent classes, there is a need to decide a proper sequence of match.

3. The Problem Statement

Anonymous union and intersection classes of an ontology may not take part in the whole inference process of a reasoner. Treating union class as super/equivalent class of its collection classes and intersection class as subclass of its collection classes produces error prone match results. They produce error prone subclass and equivalent classes too. This will affect the matching process wherein subclasses list or equivalent classes list are used. Action is needed to (a) add named union and intersection classes in the ontology, (b) interpret union and intersection classes in their original form, (c) relate union and intersection classes properly to their collection classes, and (d) avoid error prone match results of union, intersection, sub-, super- and complement classes. One can easily understand the problem statement with respect to the examples of the conference management system, as shown in Figures 1 and 2.

4. Our Proposed Algorithm

The ontology resource discovery is a process of matching requested and advertised resources. As part of the ontology resource discovery, we need to match the classes of
### Table 1: Comparison of jena query engine and Pellet query engine.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Jena query engine</th>
<th>Pellet query engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of queries dealt with</td>
<td>RDF triple based, so it works one triple at a time</td>
<td>It considers the entire conjunctive query</td>
</tr>
<tr>
<td>Speed of results</td>
<td>Query optimizations are not accessible.</td>
<td>Query optimizations are accessible and perform optimizations too and therefore can speed up the query result</td>
</tr>
<tr>
<td>Blank nodes/anonymous resources</td>
<td>For anonymous resource, it will not produce result</td>
<td>For anonymous resource, it will produce result</td>
</tr>
</tbody>
</table>

### Table 2: Comparison of interpretation of union and intersection classes by protégé vs. Pellet.

<table>
<thead>
<tr>
<th>Class</th>
<th>Interpretation by Protégé ontology editor</th>
<th>Pellet reasoner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection class</td>
<td>It adds intersection class as error prone of collection classes</td>
<td>It interprets intersection class as error prone of collection classes</td>
</tr>
<tr>
<td>Union class</td>
<td>It treats union class as equivalent of collection classes</td>
<td>It adds union class as super class of collection classes</td>
</tr>
</tbody>
</table>

**Figure 1: Union class example.**

**Figure 2: Union class example.**

ontology. Therefore, the scope of our work is ontology matching/resource discovery.

We propose a match algorithm that operates on the named union and intersection classes. We are aiming to

1. interpret the named union and intersection classes in their original form,
2. relate the union and intersection classes properly to their collection classes,
3. create a proper match sequence to avoid error prone results for union, intersection, sub-, complement, and equivalent classes.

We implement our work using two algorithms, (i) subclasses calculation algorithm and (ii) a match algorithm. In this section, we describe the algorithms which we employ and their logic of working and analyze their complexities.

### 4.1. The Logic.

The algorithms work on the following logic principles.

1. The subclass calculation algorithm is as follows.

(a) In ontology, we have $C$ as union class of $A$ and $B$ collection classes; that is,

$$ C = A \cup B \Rightarrow C \supseteq A; \quad C \supseteq B. \quad (8) $$

Consider the following.

(i) If union class is the required class and its collection class as the advertised class, the match between them will be of type "subsume" as per (4).

(ii) The Pellet reasoner adds collection classes $A$ and $B$ as the subclasses of $C$. Therefore, we have to remove collection classes from subclasses list of $C$.

Now, if

$$ D = \text{subclassOf} \ (B) \Rightarrow D > B, \quad (9) $$

from (8) and (9), $B$ is smaller than both $C$ and $D$, but we cannot comment on the relation between $C$ and $D$. Therefore, we have to remove subclasses of $B$ from the subclasses list of union class. That is, we have to remove sub classes of collection classes from the subclasses list of $C$ generated by Pellet reasoner.

(iii) The protégé editor interprets collection classes $A$ and $B$ as the equivalent classes of $C$. This can be dealt with using two ways: (i) we should remove the $A$ and $B$ from the equivalent classes list or (ii) we decide the sequence of match in a way that error prone equivalent classes list will not generate error prone match results. We match equivalent classes in the last step to avoid creating error prone match results. Resource discovery algorithm typically uses the Pellet-like reasoner, and usage of the protégé editor is not guaranteed.
Therefore, instead of removing $A$ and $B$ classes from the equivalent classes list, a proper sequence to deal with error prone equivalent classes is more preferable, we believe.

(b) In class hierarchy, if we have $C$ as intersection class

$$C = A \cap B \implies C = A; \quad C = B. \quad (10)$$

Consider the following.

(i) If intersection class is the required class and collection classes are advertised classes, the match between them will be of type "plug-in" as per (5).

(ii) The Pellet reasoner adds $C$ as the subclass of collection classes. Therefore, we have to remove $C$ from the subclasses of collection classes.

Now, if

$$D = \text{subclassOf}(C) \implies C < D, \quad (11)$$

from (10) and (11) $C$ is smaller than both $B$ and $D$, but we cannot comment on the relation between $B$ and $D$. Therefore, we have to remove subclasses of $C$ from the subclasses list of collection classes generated by Pellet reasoner.

(2) The UnionIntersectionMatch algorithm. Our match algorithm matches the complement, union, intersection, sub-, equality, and equivalent classes in a strict sequence. The change in sequence may produce error prone match results, as discussed in The Logic subsection. We apply our subclasses calculation algorithm to the complement checking algorithm of [43]. It is displayed as $\text{isComplementOf}$ function in the pseudo code section.

4.2. Pseudo Code of Our Algorithm. The pseudo code of our algorithms is as shown in Algorithms 1 and 2.

4.3. Theoretical Analysis of Our Algorithm. As our algorithms use simple loops, the worst case time complexity of our algorithm is $O(n)$.

5. Methodology of Implementation

The implementation details of our proposed algorithm are as follows.

5.1. Tools Used

(i) Fedora (version 19): Fedora [44] is an open source application platform. It offers deployment of standalone applications, web servers, virtual servers, and so forth.

(ii) Java Language (JDK1.7): Java [45, 46] language is simple, object oriented, robust, secured, architecture neutral, portable, interpreted, threaded, and dynamic, which enables high performance applications on multiple platforms in distributed environment.

(iii) Jena semantic framework: jena [18] is a Java framework for developing semantic web applications. It provides good programmatic environment for RDF (Resource Description Framework), RDFS (RDF schema), and OWL. It includes SPARQL (The Simple Protocol and RDF Query Language) and a rule-based inference engine. The jena framework includes a RDF API that reads/writes RDF in RDF/XML, N3, and N-Triples.

(iv) Pellet reasoner: the Pellet is an OWL 2 reasoner. It provides standard and cutting-edge reasoning services for OWL ontologies [20].

(v) Prot´eg´e: The prot´eg´e is an ontology editor. The support for Java and being open source make the prot´eg´e the popular reasoner [21, 22].

(vi) Netbeans (version 8.0): The NetBeans IDE [47] is a platform independent and open source IDE for Java, PHP, Ruby and Ruby on Rails, Groovy and Grails and C/C++, JavaScript, and so forth languages. It enables developers to rapidly create web, enterprise, desktop, and mobile applications and distributed applications. Its support of suggesting files import on inserting jar files is very much useful to the new developer from the jar file.

(vii) Apache web server: the apache web server [48] is an open source web server which is used here to host the ontology and service resources.

5.2. Implementation Details. Our implementation uses Linux Operating System, Java Programming Language, and Pellet reasoner. The implementation details are as follows.

(i) We add named union and intersection classes in the conference ontology [23] such that collection classes are not added as equivalent classes or subclasses of union class. They are as shown in Figure 4.

(ii) We calculate subclasses of each class in ontology, using added union and intersection classes.

(iii) We compare the results of calculating subclasses and match algorithm with our candidate reasoner, that is, the Pellet reasoner. Our algorithm uses the Pellet reasoner’s output and rectifies that. Therefore, it should take more time to calculate subclasses.

We consider mainly four cases of adding mix of intersection and union classes based on their location, as shown in Figure 3.

(i) Case 1: union class as start and end classes and intersection class in between in class hierarchy.

(ii) Case 2: union class as start and intersection class as end class in class hierarchy.
Input: $AD$ is advertised ontology dataset and $R$ is required ontology. Ontologies are described using individuals, classes, attributes/properties, relations, restrictions, rules and axioms like components. Ontology is in the format $A[\text{Classes, Properties}]$ and $R[\text{classes, properties}]

Output: Matching type of two classes of Ontology

Assumption: Ontology dataset is homogenous

1. Ontology class match _algo_
   - $obA$-advertisement and $obR$-Requested resource concepts

UnionIntersectionMatch

2. function UnionIntersectionMatchClasses($obA, obR$)
3. $match =$ "no-match"
4. if ($match = isComplementOf(obA, obR)) != "no-match" then return $match$
5. else if ($match = isUnionOf(obA, obR)) != "no-match" then return $match$
6. else if ($match = isIntersectionOf(obA, obR)) != "no-match") then return $match$
   - plug-in match
7. else if calculatesubclasses($obR$).contains($obA$) then return "plug-in"
   - subsume match
8. else if calculatesubclasses($obA$).contains($obR$) then return "subsume"
9. else if $obA.equals(obR)$ then return (1)
10. else return -(1)
11. end if
12. end function

isUnionOf

13. function isUnionOf($obA, obR$)
14. if $obA.isUnionClass()$ then
15.   $operands = obR.getOperands()$
16.   if $operands.contains(obR)$ then return "plug-in"
17.   else if calculatesubclasses($obA$).contains($obR$) then return "subsume"
18.   end if
19.   else if $obR.isUnionClass()$ then
20.     $operands = obR.getOperands()$
21.     if $operands.contains(obA)$ then return "subsume"
22.     else if calculatesubclasses($obA$).contains($obA$) then return "plug-in"
23.     end if
24.     end if
25.     end if
26.   end if
27.   return "no-match"
28. end function

isComplementOf

29. function isComplementOf($obA, obR$)
30. if $obA.isComplementClass()$ then
31.   $operands = obA.getOperands()$
32.   if $operands.contains(obR)$ then return "complement"
33.   else if calculatesubclasses($operands$).contains($obR$) then return "complement"
34.   end if
35.   end if
36. else if $obR.isComplementClass()$ then
37.   $operands = obR.getOperands()$
38.   if $operands.contains(obA)$ then return "complement"
39.   else if calculatesubclasses($operands$).contains($obA$) then return "complement"
40.   end if
41.   end if
42. end function

isIntersectionOf

43. function isIntersectionOf($obA, obR$)
44. if $obA.isIntersectionClass()$ then

Algorithm 1: Continued.
8 TheScientificWorldJournal

Algorithm 1: Pseudo code of our UnionIntersectionMatch algorithm.

```plaintext
operands = obA.getOperands()
if operands.contains(obR) or calculatesubclasses(operands).contains(obR) then return "subsume";
else if obR.isIntersectionClass() then
    operands = obA.getOperands()
    if operands.contains(obA) or calculatesubclasses(operands).contains(obA) then return "plug-in"
else return "no-match"
end if
end function
```

Algorithm 2: Pseudo code of our Calculated subclasses algorithm.

```plaintext
Function calculatesubclasses
(1) function calculatesubclasses(cls, unionClassesList, intersectionClassesList)
▷ Use output of Pellet reasoner to as input to our algo and modify that
(2) subclassesSet = subClassesOf(cls)
▷ Separate the intersection and union classes from sub classes list
(3) Un = intersectionOf(unionClassesList, subclassesSet)
(4) In = intersectionOf(intersectionClassesList, subclassesSet)
(5) if obR isUnionClass then
    Un = Un + obR
(6) for i in In do
    ▷ Get collection classes
    operands = getOperands(u)
    if obR == operands then
        subclassesSet-i ▷ Remove sub classes of intersection class, as they too are error prone
        subclassesSet-i.listsubclasses()
        ▷ Assumption: a class cannot be both sub class and intersection class of collection classes
        if collection class and intersection class both are added in subclass list, remove intersection class
        else if subclassesSet.contains(operands) & subclassesSet.contains(i) then
            subclassesSet-i
            ▷ Remove sub classes of intersection class, as they too are error prone
            subclassesSet-i.listsubclasses()
            end if
    end for
    ▷ remove error prone operands of union class as sub classes
    for u in Un do
        operands = getOperands(u)
        ▷
        subclassesSet = subclassesSet-operands-subClassesOf(operands)
    end for
    ▷ Jena entails a class itself as the subclass of its own, so need to remove union class from sub classes list
    subclassesSet = subclassesSet-cls return subclassesSet
(23) end function
```

(iii) Case 3: intersection class as start and union class as end in class hierarchy.

(iv) Case 4: intersection class as start and end classes and union class in between in class hierarchy.

We add the above mentioned cases in the conference management system’s ontology. It is shown in Figure 4.

5.3. Test Application. As the test application, we use the conference management system, because we believe the standard data set of conference ontology [23] is nowadays used to test the match algorithms.

5.4. Evaluation Metrics. We use precision ratio as the evaluation parameter.
6. Performance Results & Analysis

We have added the named union classes, intersection classes, and subclasses in an ontology with four different cases as shown in Figures 3 and 4. We compare the results of our subclasses calculating algorithm with those of the Pellet reasoner. From Tables 3, 4, 5, and 6, we can say, the execution time of our algorithm is more than that of the Pellet reasoner. The precision ratio of our subclass calculating algorithm increases from 70% to 100%. Though the execution time of our algorithm is more, it gives better precision ratio. We can ignore investing few more milliseconds of time, to get ethically correct interpretations of union and intersection classes, with increased precision ratio. The real world conference ontology with our added classes and cases results are as shown in Tables 7, 8, 9, and 10. We are able to get the corrected subclasses list using our algorithm.

For general cases, our algorithm improves the precision ratio, as shown in Figure 5. For the conferences cases, the improved ratio is as shown in Figure 6. For UnionIntersectionMatch algorithm the performance results are as shown in Table II. We can observe that the relation of union and intersection classes with their collection classes is not “exact match,” but rather subsume or plug-in match. The results obtained are different than those we get from the Pellet reasoner. The changes in the results are shown highlighted in Table II.

7. Conclusion and Future Work

A resource discovery is a challenging task, because of the unlimited size of the semantic web. The ontologies available in literature typically contain anonymous union and intersection classes. The anonymous classes may not take part in the whole inference process of a reasoner. The candidate ontology editor, the protége [21, 22], interprets the named union and its collection classes as equivalent classes. The candidate
Table 4: Calculating list of subclasses: case 2.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Subclasses by Pellet</th>
<th>Time by Pellet (ms)</th>
<th>Subclasses by our algo.</th>
<th>Time by our algo. (ms)</th>
<th>Precision by Pellet</th>
<th>Precision by our algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>[B, C, and D]</td>
<td>9</td>
<td>[]</td>
<td>45</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>[D]</td>
<td>29</td>
<td>[]</td>
<td>49</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>[D]</td>
<td>0</td>
<td>[]</td>
<td>18</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>[]</td>
<td>0</td>
<td>[]</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 5: Calculating list of subclasses: case 3.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Subclasses by Pellet</th>
<th>Time by Pellet (ms)</th>
<th>Subclasses by our algo.</th>
<th>Time by our algo. (ms)</th>
<th>Precision by Pellet</th>
<th>Precision by our algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>[C, D, E, and F]</td>
<td>35</td>
<td>[]</td>
<td>83</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>[C, D, E, and F]</td>
<td>1</td>
<td>[]</td>
<td>44</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>[D, E, and F]</td>
<td>1</td>
<td>[D]</td>
<td>24</td>
<td>0.33</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>[E and F]</td>
<td>1</td>
<td>[]</td>
<td>19</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>[]</td>
<td>0</td>
<td>[]</td>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>[]</td>
<td>0</td>
<td>[]</td>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 6: Calculating list of subclasses: case 4.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Subclasses by Pellet</th>
<th>Time by Pellet (ms)</th>
<th>Subclasses by our algo.</th>
<th>Time by our algo. (ms)</th>
<th>Precision by Pellet</th>
<th>Precision by our algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>[]</td>
<td>0</td>
<td>[]</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>[A, D, E, F, and G]</td>
<td>1</td>
<td>[D and G]</td>
<td>94</td>
<td>0.40</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>[A]</td>
<td>0</td>
<td>[]</td>
<td>21</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>[E, F, and G]</td>
<td>1</td>
<td>[G]</td>
<td>35</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>[]</td>
<td>15</td>
<td>[]</td>
<td>15</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>[]</td>
<td>1</td>
<td>[]</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
<td>[]</td>
<td>53</td>
<td>[]</td>
<td>55</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>H</td>
<td>[G]</td>
<td>0</td>
<td>[]</td>
<td>22</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>[G]</td>
<td>0</td>
<td>[]</td>
<td>16</td>
<td>0.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 7: Conference cases subclass calculation results: case 1.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Pellet subclasses</th>
<th>Our algo. subclasses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chief guest</td>
<td>[Sponsor, KnowledgeablePerson, MainSponsor, AudioVideoSponsor, AudioSponsor, and VideoSponsor]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>Sponsor</td>
<td>[MainSponsor, AudioVideoSponsor, AudioSponsor, and VideoSponsor]</td>
<td>[MainSponsor and AudioVideoSponsor]</td>
</tr>
<tr>
<td>3</td>
<td>KnowledgeablePerson</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>4</td>
<td>MainSponsor</td>
<td>[AudioVideoSponsor, AudioSponsor, and VideoSponsor]</td>
<td>[AudioVideoSponsor]</td>
</tr>
<tr>
<td>5</td>
<td>RecommendedSponsorByDirector</td>
<td>[MainSponsor, AudioVideoSponsor, AudioSponsor, and VideoSponsor]</td>
<td>[]</td>
</tr>
<tr>
<td>6</td>
<td>ReputedSponsor</td>
<td>[MainSponsor, AudioVideoSponsor, AudioSponsor, and VideoSponsor]</td>
<td>[]</td>
</tr>
<tr>
<td>7</td>
<td>AudioVideoSponsor</td>
<td>[AudioVideoSponsor]</td>
<td>[]</td>
</tr>
<tr>
<td>8</td>
<td>AudioSponsor</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>9</td>
<td>VideoSponsor</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>
Case 1
Entry and exit with union, in between subclass and intersection

Case 2
Entry with intersection, in between subclass and ends with union

Case 3
Entry with union, and ends with intersection

Case 4
Entry and exit with intersection, in between subclass and union

Figure 4: Cases of adding union and intersection classes in conference ontology.

Table 8: Conference cases subclass calculation results: case 2.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Subclasses by Pellet</th>
<th>Subclasses by our algo.</th>
<th>Precision by Pellet</th>
<th>Precision by our algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Speaker</td>
<td>[AcademicExpert, IndustryExpert, and Scientist]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>AcademicExpert</td>
<td>[Scientist]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>IndustryExpert</td>
<td>[Scientist]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>Scientist</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 9: Conference cases subclass calculation results: case 3.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Subclasses by Pellet</th>
<th>Subclasses by our algo.</th>
<th>Precision by Pellet</th>
<th>Precision by our algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WellWrittenPapers</td>
<td>[AcceptedPapers, PapersPublishedInJournal, ExcellentPapers, and WellPresentedPaper]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>GoodTechnicalPapers</td>
<td>[AcceptedPapers, PapersPublishedInJournal, ExcellentPapers, WellPresentedPaper]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>AcceptedPapers</td>
<td>[PapersPublishedInJournal, ExcellentPapers, and WellPresentedPaper]</td>
<td>[PapersPublishedInJournal]</td>
<td>0.33</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>PapersPublishedInJournal</td>
<td>[ExcellentPapers and WellPresentedPaper]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>ExcellentPapers</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>WellPresentedPaper</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Table 10: Conference cases subclass calculation results: case 4.

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Subclasses by Pellet</th>
<th>Subclasses by our algo.</th>
<th>Precision by Pellet</th>
<th>Precision by our algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scientist</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>AcademicExpert</td>
<td>[Scientist, ComputerExpert, SecurityExpert, SemanticWebExpert, and OSExpert]</td>
<td>[ComputerExpert and OSExpert]</td>
<td>0.40</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>IndustryExpert</td>
<td>[Scientist]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>ComputerExpert</td>
<td>[SecurityExpert, SemanticWebExpert, and OSExpert]</td>
<td>[OSExpert]</td>
<td>0.33</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>SecurityExpert</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>SemanticWebExpert</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>OSExpert</td>
<td>[]</td>
<td>[]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>CommandExpert</td>
<td>[OSExpert]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>KernelExpert</td>
<td>[OSExpert]</td>
<td>[]</td>
<td>0.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 5: Subclass calculation precision ratio by Pellet and our algo.: cases 1, 2, 3, and 4.
reasoner, namely, Pellet [20, 41, 49] adds the intersection class as subclass of its collection classes, whereas it adds union class as superclass of its collection classes. Therefore, the relations between collection classes with their intersection and union class are error prone. In addition, the subclasses list inferred by the Pellet also can be error prone. Therefore, in this paper we have proposed algorithms to (a) relate intersection and union classes correctly to their collection classes, (b) calculate correct subclasses list of any class of ontology, wherein the class hierarchy contains intersection and union classes, and (c) match the required and advertised classes using proper sequence of the above two steps with complement
classes added. We have compared our subclasses calculation algorithm results with those of our candidate reasoner, that is, the Pellet reasoner. Though the execution time of our subclasses calculation algorithm is more than that of the Pellet reasoner, we get improved precision ratio from 70% to 100% for our ontology cases. The UnionIntersectionMatch algorithm has different match results than those of the Pellet reasoner. Our match algorithm must follow the steps in the specified sequence of (complement, union, intersection, sub-., equality, and equivalent classes) ONLY, or else it may produce error prone results. To the best of our knowledge, ours is a unique and novel way to relate ontology classes.

As a future work, one can find proper subclasses of anonymous union and intersection classes of existing ontologies. One can propose subproperties inclusion in the algorithm. One can use our finally improved algorithm with content based similarity measure of Signature Quadratic Form Distance (SQFD) [50] to find alignment between two ontologies. Our algorithm, if used in ontology alignment algorithm, will improve precision ratio of resource discovery, we believe.

Conflict of Interests

There is no conflict of interests regarding the publication of this paper.

References

Submit your manuscripts at
http://www.hindawi.com