Checking and Handling Inconsistency of DBpedia

Zhaohua Sheng, Xin Wang*, Hong Shi, and Zhiyong Feng

School of Computer Science and Technology, Tianjin University, Tianjin, China
shengzhaoli24353@sina.com, {wangx, serena, zyfeng}@tju.edu.cn

Abstract. DBpedia is the hub of Linked Data, and there might be inconsistencies in it. Reasoning with inconsistent ontologies may lead to erroneous conclusions, so whether it is consistent is a critical issue. However, the current inference engine is only appropriate to reason lightweight ontologies, and the existing approaches to handle inconsistencies are unreasonable. In this paper, we check the inconsistency in DBpedia by rule-based distributed reasoning using MapReduce. The experimental results show that there are a number of inconsistencies in DBpedia. Furthermore, we should handle different types of inconsistencies respectively with different methods to improve data quality of DBpedia.

Keywords: inconsistency, large-scale ontology, DBpedia, MapReduce.

1 Introduction

The DBpedia project [1] is a community effort to publish data of Wikipedia infobox in the form of RDF. It has large volume of data and is relatively comprehensive. During the last few years, a great number of Linked Data datasets have been published in which entities are linked to their equivalent resource in DBpedia, making DBpedia a central interlinking hub for the Linked Data [2] datasets.

There might be inconsistencies in DBpedia. From data sources viewpoint, one's understanding of something may vary according to his own background. So, inconsistent data might exist in Wikipedia. From another viewpoint, when the data are extracted from Wikipedia, errors may occur during this process. For example, when we trace LibriVox to its Wikipedia page, we find that it is in fact an online digital library. However, in DBpedia it is asserted as a member of libraries in the real world. Reasoning with inconsistent ontologies may lead to erroneous conclusions. Hence, we need to check its consistency and handle inconsistencies with a reasonable method.

Since DBpedia is the hub of Linked Data and there are serious consequences if reasoning with inconsistent premises, we focus on checking and handling inconsistency of it in this paper. However, the TBox of the DBpedia ontology is relatively simple such that we cannot directly check its consistency. So, firstly we extend the DBpedia ontology with disjointness axioms. Ontologies, or generally speaking schemas, are an effective mean to improve the quality of Linked Data.

* Corresponding author.

F.L. Wang et al. (Eds.): WISM 2012, LNCS 7529, pp. 480–488, 2012.

In particular, the disjointness axioms are definitions of relation between two classes that share no instances, and can be used to check the consistency of ontologies. Thus, enriching ontologies with disjointness axioms is a hot topic in ontology engineering.

The remainder of this paper is organized as follows: after giving a brief overview of related work in Section 2, we present how to extend the DBpedia ontology using UMBEL in Section 3. Based on this, we show the consistency checking of DBpedia in detail in Section 4. In Section 5 we present our analysis and discussion of experimental results. Finally, there is our conclusion and future work in Section 6.

2 Related Work

The consistency is an important metric for quality of ontologies. Many researchers have been working on checking and handling inconsistencies of large-scale RDF data.

In [3] they define some types of inconsistency when measuring data quality of Linked Data. But they only give representations of inconsistencies in the natural language and there are no formal definitions. Pellet [4], one of the most popular OWL reasoners, could check the consistency of ontologies. However, Pellet is only appropriate to reason lightweight ontologies and there are memory overflow problems when reasoning large-scale ontologies. It is required to use a distributed method to solve this problem. WebPIE [5], proposed by J. Urbani, is a distributed inference engine based on MapReduce [6]. Its experimental results show that it has high efficiency and good scalability. But it only restricts to RDFS and OWL Horst rules.

When handling inconsistencies, the traditional approach is just removing or ignoring the corresponding axioms uniformly. In [7][8] they make efforts to obtain a collection of maximum consistent sub-ontologies. Both of the approaches have a shortcoming that they lack considering reasons causing inconsistencies.

The work in [9] is similar to ours. But it has two drawbacks. One is that they convert consistency checking to SPARQL query, so they do not tackle the essential large-scale ontology reasoning problem. The other is that they just simply list the number of inconsistencies in DBpedia, but they do not analyze the experimental results thoroughly and give no reasonable solutions for handling inconsistencies.

3 Ontology Extension of DBpedia

As shown in [1], the TBox of DBpedia is relatively simple such that we cannot directly check its consistencies. So, firstly we extend it with disjointness axioms using the relation between UMBEL and DBpedia.

UMBEL [10] (Upper Mapping and Binding Exchange Layer) is a lightweight ontology which has been created for interlinking Web content and data. It is a subset of OpenCyc[11]. OpenCyc is a common knowledge base and has a long development history. Now, it has been accepted around the world and there are many applications based on it. So, there is a strong persuasiveness to extend the DBpedia ontology using UMBEL. UMBEL has two valuable functions. First, it defines a base vocabulary which can be used for the construction of ontologies from other domains. Second,

based on the vocabulary, it provides many reference concepts that can be used as standard links in external datasets, so different datasets could interoperate easily.

There are two reasons why we can use UMBEL to obtain disjointness axioms in the DBpedia ontology. First, the UMBEL ontology defines the disjointness relations between different classes. Second, the DBpedia ontology is linked to the UMBEL ontology by rdf:subClassOf. Thus, we could extend the DBpedia ontology using rule (1) and obtain 25534 disjointness axioms.

$$\frac{(RC1, \text{rdfs:subClassOf}, C1)(RC2, \text{rdfs:subClassOf}, C2)(C1, \text{owl:disjointWith}, C2)}{(RC1, \text{owl:disjointWith}, RC2)} \quad (1)$$

Note that RC_1 and RC_2 are classes from DBpedia and C_1 and C_2 are classes from UMBEL. All relationships are defined in files downloaded from the website[1].

4 Reasoning with MapReduce to Check Inconsistency

In this Section, we first give a brief introduction to MapReduce. Next, we summarize five types of inconsistencies in the form of inference rules in DBpedia based on [3]. Finally, we take a rule as an example to describe how we implement it using MapReduce.

4.1 The MapReduce Programming Model

The MapReduce programming model proposed by Google is a parallel processing model and suitable for handling large-scale data. A cluster of some commodity machines could achieve high efficiency and scalability using this model. In recent years, there is a lot of work using this model when consuming Linked Data.

Each job consists of two phases: a Map phrase and a Reduce phrase. When a job arrives, the input files are divided into many chunks. Then each chunk is processed respectively. The Map phase emits the input data in the form of a set of key/value pairs. All pairs with the same key will be collected to the same machine. Every machine processes intermediate values independently and outputs a new set of key/value pairs as output.

4.2 Definitions of Inconsistencies in DBpedia

INCONSISTENCY 1: Usage of undefined classes or properties

$$\frac{(s, \text{rdf:type}, X)}{(\text{inconsistency})} \quad (X \text{ is not defined in TBox.}) \quad (2)$$

$$\frac{(s, p, o)}{(\text{inconsistency})} \quad (p \text{ is not defined in TBox.}) \quad (3)$$

[1] http://code.google.com/p/umbel/source/browse/#svn%2Ftrunk%2Fv100

INCONSISTENCY 2: Usage of literals incompatible with ranges of data type properties

$$\frac{(p, \text{rdfs:range}, d_1)(s, p, o)(o, \text{rdf:type}, d_2)}{(\text{inconsistency})} \quad (d_1 \text{ is incompatible with } d_2.) \tag{4}$$

INCONSISTENCY 3: One class is subclass of and disjoint with another class

$$\frac{(X, \text{owl:disjointWith}, Y)(X, \text{rdfs:subClassOf}, Y)}{(\text{inconsistency})} \tag{5}$$

INCONSISTENCY 4: One class is subclass of two disjoint classes

$$\frac{(X, \text{rdfs:subClassOf}, Y)(X, \text{rdfs:subClassOf}, Z)(Y, \text{owl:disjointWith}, Z)}{(\text{inconsistency})} \tag{6}$$

INCONSISTENCY 5: Invalid entity definitions as members of disjoint classes

$$\frac{(s, \text{rdf:type}, X)(s, \text{rdf:type}, Y)(X, \text{owl:disjointWith}, Y)}{(\text{inconsistency})} \tag{7}$$

Note that s and o stand for instances; p stands for properties defined in the TBox; X, Y and Z stand for classes defined in TBox; d_1 and d_2 stand for data types from XML or DBpedia.

4.3 Example Rule Execution with MapReduce

In this Section, we take rule (7) as an example to illustrate how the inference rule is executed with MapReduce. The inputs of this job are documents of triples in the form (s, p, o). In the Map phrase, triples with rdf:type as predicates and classes as objects are emitted. The intermediate outputs are pairs of <s, o> as shown in Algorithm 1 and Fig. 1. After the Map phrase, values with the same key are collected to one machine. When all emitted triples are grouped for the reduce phase. The reduce function starts to work. In the Reduce phrase, we traverse values to check whether the two classes are disjoint. If they are, an inconsistency is output; if not, nothing is output.

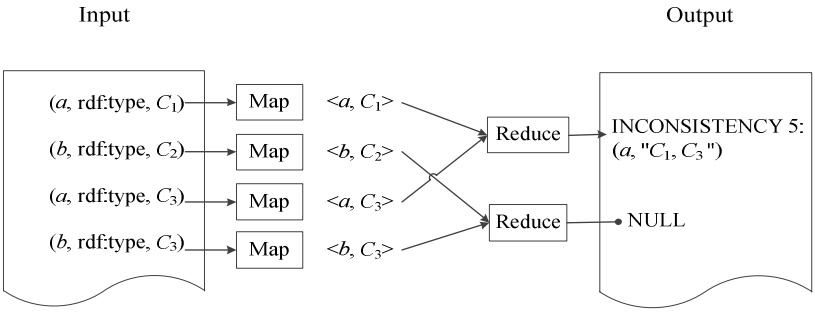

Fig. 1. Encoding rule (7) with MapReduce. C_1 and C_3 are disjoint classes; C_2 and C_3 are not

Algorithm 1. Encoding rule 7 with MapReduce

```
1: map (key, value)
        // key: line number (not relevent).
        // value:triples.
2:   if (value.p=rdf:type and value.o=owl:Class)
3:       EMIT (<value.s, value.o>)

1: reduce (key, iterator values)
        // key: resource.
        // values: classes which are types of the resource.
2:   for each v1 in values
3:     for each v2 in values
4:       if (v1 owl:disjointWith v2)
5:         new INCONSISTENCT 5 (key, "v1, v2")
6:         EMIT (<NullWritable, INCONCSISTENCY 5>)
```

5 Experiments

We conduct several experiments on the dataset provided by DBpedia[2]. So far, the latest version is 3.7. First we check consistencies in the original dataset, and then we experiment on the dataset enriched with triples obtained according to two RDFS reasoning rules (8, 9). Since RDF data have semantics and we often consume inferred data, it is necessary to check consistencies of it after reasoning.

In this section, we use the following prefixes to abbreviate resource URIs in the DBpedia dataset.

dbo: http://dbpedia.org/ontology.
dbr: http://dbpedia.org/resource.
dbd: http://dbpedia.org/datatype.

5.1 Data Inconsistency before Reasoning

(1) Usage of undefined classes or properties

***Experimental Results*:** In DBpedia, there are no classes used without definitions. But 15 undefined properties are used in the ABox which consists of foaf:based_near, foaf:familyName, foaf:surname, foaf:page, foaf:thumbnail, foaf:homepage, foaf:logo, foaf:name, foaf:givenName, foaf:nick, geo:lat, geo:long, skos:subject, purl:description, and georss:point. The number of triples using undefined properties is 3170567 and accounts for 18% of all.

***Discussion*:** From the result we conclude that although a lot of undefined properties are used, they are just not defined in DBpedia. These properties are from FOAF, the geospatial namespace, SKOS, Dublin Core, and GeoRSS.

[2] http://wiki.dbpedia.org/Downloads37

***Solution*:** We need to import ontologies which define these properties in the header of the DBpedia document.

(2) Usage of literals incompatible with ranges of data type properties

***Experimental results*:** There are 39632 triples whose literals are incompatible with ranges of data type properties. Table 1 enumerates the top five properties instantiated with incompatible values.

***Discussion*:** We find that ranges of these properties are all defined to xsd:double in the TBox. However, in the ABox, objects in triples with these properties are assigned to user-defined data types in DBpedia and they are not of the same type. So, there is no proper and unified data type for this type of properties.

***Solution*:** There are two solutions to solve this problem. First, we could make no definition for its range and the value of it can be of any data types. Second, we could define a new data type as superclass of these existing types as its range in the TBox. In the ABox, we could assert its value to the corresponding subclasses if necessary.

Table 1. Top five data type properties with literals incompatible with ranges

Data type properties	Number of occurrences
<http://dbpedia.org/ontology/budget>	8248
<http://dbpedia.org/ontology/revenue>	6960
<http://dbpedia.org/ontology/gross>	6447
<http://dbpedia.org/ontology/netIncome>	3502
<http://dbpedia.org/ontology/cost>	2871

(3) One class is subclass of two disjoint classes

***Experimental results*:** We find the following inconsistency of this type.
(dbo:Library , rdfs:subClassOf , dbo:Building)
(dbo:Library, rdfs:subClassOf, dbo:EducationalInstitution)
(dbo:Building , owl:disjointWith, dbo:EducationalInstitution)

***Discussion*:** When analyzed respectively they are all correct because of various characteristics of something. But there are real world libraries and digital ones. When it is the latter, the first axiom is wrong. Therefore, it is unreasonable to decide which of them should be removed only from the concept definitions.

***Solution*:** Classes defined in the DBpedia TBox are not comprehensive and the granularity of classification is relatively rough. We should improve quality of the DBpedia TBox. For example, we could define subclasses of the class dbo:Library.

(4) Invalid entity definitions as members of disjoint classes

***Experimental results*:** We find 960 inconsistencies of this type. All these inconsistencies are instances of the following two disjointness axioms.

(dbo:Building, owl:disjointWith, dbo:EducationalInstitution)
(dbo:Building, owl:disjointWith, dbo:Organisation)

***Discussion*:** As shown in Section 1, there are some reasons that cause inconsistencies. In order to handle inconsistencies reasonably we need to seek out what causes contradictions on earth. The first axiom is reasonable since we believe that buildings are disjoint with organizations. For example, we can assert dbr: Yale_University_Library to an instance of dbo:Building and dbo:Organization. Therefore, when handling this inconsistency we should ignore the disjointness axiom. But when we consider dbr:LibriVox the solution is not proper. It is in fact a digital library. There is an erroneous assertion when data are extracted from Wikipedia. So when handling this inconsistency we should remove the incorrect assertion.

***Solution*:** We divide this type of inconsistency into two categories based on reasons: (1) Inconsistencies that should be retained due to various characteristics of things. (2) Inconsistencies that should be removed. We should deal with each inconsistency respectively according to its category.

5.2 Data Inconsistency after Reasoning

As triples we obtain only relate to inconsistency 5, other experimental results are the same to results before reasoning. In this Section, we only show results of inconsistency 5.

***Experimental results*:** We obtain 2,483,391 triples with rules (8, 9) and find 55829 inconsistencies. Table 2 enumerates the top five disjoint classes.

***Discussion*:** The number and type of inconsistencies both increase after reasoning. As shown in Table 2, the top two pairs of disjoint classes are the same as results before reasoning. In addition, we also find apparent inconsistencies such as dbo:PopulatedPlace and dbo:Person. The results indicate that DBpedia cannot guarantee the correctness of reasoning. In other words, there are a lot of contradictions between the TBox and the ABox. The essential reason is that the domain or range of one property in the TBox is disjoint with types of an instance in the ABox.

***Solution*:** If we would like to develop applications based on the DBpedia dataset we should pre-process the data first of all. Fortunately, our results point out the direction in some extent. For example, we recommend to analyzing instances asserted to classes in Table 2.

$$\frac{(p, \text{rdfs:domain}, X)(s, p, o)}{(s, \text{rdf:type}, X)} \tag{8}$$

$$\frac{(p, \text{rdfs:range}, X)(s, p, o)}{(o, \text{rdf:type}, X)} \tag{9}$$

Table 2. Top five disjoint classes

Disjoint classes	Number of instances
Organisation & Building	10030
EducationalInstitution & Building	8559
School & Building	7910
MeanOfTransportation & Fungus	7692
PopulatedPlace & Person	1473

6 Conclusion and Future Work

In this paper, we focus on checking consistencies of DBpedia. We first extend it with disjointness axioms via UMBEL. Then we define five types of inconsistencies and present them in the form of six inference rules. Based on that, an algorithm using MapReduce is proposed to check consistencies for large-scale ontologies. After analyzing the experimental results we summarize three reasons that cause inconsistencies in DBpedia: (1) user-defined data types of DBpedia are inadequate; (2) there are a lot of incorrect assertions in the DBpedia ABox; and (3) the quality of the DBpedia TBox is low. For instance, the granularity of classification is relatively rough. Finally, we draw the conclusion that it is not wise just to remove or ignore corresponding axioms uniformly and we should classify inconsistencies according to the reasons and deal with them respectively.

In the future, we will give more rules of inconsistencies for more datasets such as GeoNames and Bio2RDF. Regards to handling inconsistencies, an automatic inconsistency classifier is needed.

Acknowledgements. This work was supported by the National Science Foundation of China (No. 61100049, 61070202) and the Seed Foundation of Tianjin University (No.60302022).

References

1. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - A Crystallization Point for the Web of Data. Web Semantics 7(3), 154–165 (2009)
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
3. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the pedantic web. In: International Workshop on LDOW at WWW (2010)
4. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics 5, 51–53 (2007)
5. Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., Bal, H.: WebPIE: A Web-scale parallel inference engine using MapReduce. Web Semantics: Science, Services and Agents on the World Wide Web (2011)
6. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 107–113 (2008)

7. Haase, P., Stojanovic, L.: Consistent evolution of OWL ontologies. The Semantic Web: Research and Applications, 91–133 (2005)
8. Meyer, T., Lee, K., Booth, R., Pan, J.Z.: Finding maximally satisfiable terminologies for the description logic ALC. In: Proceedings of the 21st National Conference on Artificial Intelligence, pp. 269–274. AAAI Press (2006)
9. Fleischhacker, D., Völker, J.: Inductive Learning of Disjointness Axioms. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part II. LNCS, vol. 7045, pp. 680–697. Springer, Heidelberg (2011)
10. UMBEL, `http://www.umbel.org/`
11. OpenCyc, `http://www.opencyc.org/`