Knowledge-Based Systems 71 (2014) 169–186

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Measuring effectiveness of ontology debugging systems

Qiu Ji [a,b,*], Zhiqiang Gao [a,b], Zhisheng Huang [c], Man Zhu [a,b]

[a] School of Computer Science and Engineering, Southeast University, Nanjing, China
[b] Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, Nanjing, China
[c] Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 30 October 2013
Received in revised form 28 July 2014
Accepted 28 July 2014
Available online 12 August 2014

Keywords:
Semantic Web
Ontology
Ontology debugging
Inconsistency handling
Effectiveness measure

ABSTRACT

Ontology debugging aims to provide users with justifications for an entailment in OWL ontologies. So far, many ontology debugging algorithms have been proposed and several ontology debugging systems are available. There has been some work on evaluating these systems with the efficiency as the main evaluation measure. However, existing systems may fail to find all justifications for an entailment within a time limit and may return incorrect justifications. Therefore, measuring their effectiveness by considering the correctness of justifications and the completeness of a found set of justifications is helpful. In this paper, we first give a survey of existing ontology debugging approaches and systems. We then evaluate both the effectiveness and the efficiency of existing ontology debugging systems based on a large collection of diverse ontologies. To assess the effectiveness of an ontology debugging system, we first propose a method to construct the reference justification sets and define the degrees of correctness and completeness of the system. Then we construct a dataset containing 80 ontologies with significantly different sizes and expressivities. Based on the proposed evaluation measures and the constructed dataset, we do comprehensive experiments. The results show the advantages and disadvantages of existing ontology debugging systems in terms of correctness, completeness and efficiency. Based on the results, we provide several suggestions for users to choose an appropriate ontology debugging system and for developers to design an ontology debugging algorithm and build an ontology debugging system.

1. Introduction

It is well known that ontologies – explicit and formal specifications of the terms in a domain and the relations among them [28] – play an important role in the formal representation of knowledge on the Semantic Web. Since Web Ontology Language (OWL) [2] became a W3C recommendation standard, it has been used in many application domains such as e-Science, bio-informatics and geography [34]. Description Logics (DLs) [3], which are a family of knowledge representation languages, provide a well-defined formal semantics for OWL. They make it possible to infer implicit axioms from a set of explicit axioms, where an inferred axiom is called an *entailment*. One of the important reasoning tasks of an OWL reasoner is to check if an axiom can be entailed by an ontology consisting of a set of axioms. This task is called *entailment checking*. Two special subtasks of entailment checking are unsatisfiability checking of a concept and inconsistency checking of an ontology.[1] These reasoning tasks have been proven essential to design and maintain high-quality ontologies [84] and answer queries [37].

In practice, building ontologies is an error-prone effort and logical contradictions are always unavoidable [50,52,55,56,62,69,83]. When an ontology contains errors, some entailments of the ontology can be undesirable. In such a case, users may want to know which parts of the ontology cause the undesirable entailments and decide which axioms in the ontology can be modified. Therefore, providing explanations for the undesirable entailments is an important task of ontology engineering [78]. Ontology debugging aims to provide users with explanations of entailments in an OWL ontology, by computing justifications for the entailment. A justification is a minimal set of axioms that can explain an entailment. In real ontologies, there may exist many justifications for some entailments and the reasons for entailments may range from fairly simple localized reasons to highly non-obvious reasons [33].

* Corresponding author at: School of Computer Science and Engineering, Southeast University, Nanjing, China.
E-mail addresses: jiqiu@seu.edu.cn (Q. Ji), zqgao@seu.edu.cn (Z. Gao), huang@cs.vu.nl (Z. Huang), mzhu@seu.edu.cn (M. Zhu).

[1] A concept is unsatisfiable if it is interpreted as an empty set. An ontology is inconsistent if it has no model.

http://dx.doi.org/10.1016/j.knosys.2014.07.023
0950-7051/© 2014 Elsevier B.V. All rights reserved.

In such cases, without a system's support, it is very difficult or even infeasible to comprehend why an entailment holds.

So far, many ontology debugging algorithms have been proposed. They generally can be classified into two categories: a glass-box approach which modifies the reasoning algorithm of an OWL reasoner [73] and a black-box approach which treats the reasoner as a "black-box" or an "oracle" [51]. Through implementing different ontology debugging algorithms, various ontology debugging systems such as DION[2] and RaDON[3] have been developed. To see the advantages and disadvantages of existing ontology debugging systems, the work in [78] conducts an evaluation of five representative ontology debugging systems over a few incoherent ontologies.[4] In [33], a more extensive investigation is carried out to evaluate a black-box algorithm for finding all justifications over a collection of ontologies. A recent survey in [87] compares existing ontology debugging algorithms for finding all justifications.

Although there has been some work on evaluating existing ontology debugging systems, the efficiency is their main evaluation measure. However, these systems may fail to find all justifications for an entailment within a time limit and even return incorrect justifications. Therefore, measuring their effectiveness by considering the correctness of justifications and completeness of a set of found justifications[5] is helpful. It is well known that measuring the effectiveness of an information retrieval system has been extensively studied, where the effectiveness is used to determine the relevance of documents retrieved by a search engine [70]. Two typical measures of effectiveness in information retrieval are precision and recall [8,57] which are commonly used and well understood. The precision is the fraction of retrieved documents that are relevant and the recall is the fraction of relevant documents that have been retrieved. To measure the effectiveness of an ontology debugging system, we define two measures, i.e., the degree of correctness and the degree of completeness, which correspond to precision and recall respectively.

In this paper, we evaluate both effectiveness and efficiency of existing ontology debugging systems based on a large collection of diverse ontologies. To assess the effectiveness, we first propose a method to construct the *reference justification sets* and define the degrees of correctness and completeness of the system. Then we construct a dataset containing 80 ontologies which are not "cherry picked" for various ontology debugging tasks and vary greatly in size and expressivity. These ontologies contain a much wider range of the numbers of justifications than those provided in existing evaluation work. Based on the proposed evaluation measures and the constructed dataset, we do comprehensive experiments. The evaluation results reveal that a number of justifications found by three selected systems are incorrect, and three systems that implement an algorithm satisfying the completeness cannot always find all justifications when there is no timeout or "out of memory" error. Our evaluation results also show that, although existing ontology debugging systems have achieved good performance w.r.t. efficiency, there still exists some space for improvement.

In summary, our paper contains the following contributions:

(1) We define evaluation measures to assess the effectiveness of ontology debugging systems by adapting the notions of precision and recall.
(2) We construct a dataset for various ontology debugging tasks. First, we carefully select some representative existing ontologies. Second, we construct some new ontologies for different evaluation purposes, such as evaluating merged ontologies and testing the scalability of existing ontology debugging systems.
(3) We conduct an extensive empirical evaluation of existing ontology debugging systems to evaluate their effectiveness and efficiency. To the best of our knowledge, this is the first work that comprehensively evaluates the effectiveness of ontology debugging systems.
(4) Our evaluation facilitates users to choose an appropriate ontology debugging system and developers to design an ontology debugging algorithm and build a system.

[2] http://wasp.cs.vu.nl/sekt/dion/.
[3] http://atur.aturstudio.com/homepage/qiuji/radon.htm.
[4] An ontology is incoherent if it contains at least one unsatisfiable concept.
[5] A justification for an entailment is correct if it is a minimal axiom set to infer the entailment. A set of justifications for an entailment is complete if all justifications of the entailment are included in the set.

The rest of this paper is organized as follows. We present the background knowledge in Section 2. In Section 3, we describe the ontology debugging systems to be evaluated. We then give the evaluation measures and dataset in Section 4 and Section 5 respectively. After that, the evaluation process is introduced in Section 6 and the experimental results are given in Section 7. Based on the results, we provide suggestions to users and developers of ontology debugging systems in Section 8. Finally, we discuss related work in Section 9 and conclude this paper in Section 10.

2. Background knowledge

In this section, we first give the basics of description logics and ontology mapping. We then introduce the key notions used for debugging ontologies. Finally, we give a brief survey of existing ontology debugging approaches.

2.1. Description logics

It is known that the highly expressive DL $\mathcal{SHOIN}$ [36] underpins OWL [58] and the more expressive DL $\mathcal{SROIQ}$ [35] underpins OWL 2 [27]. As most of ontology debugging systems to be evaluated can deal with OWL 2 ontologies, we give a brief introduction of DL $\mathcal{SROIQ}$.

2.1.1. Description logic syntax

In DLs, there are three kinds of entities: concepts representing sets of individuals, roles representing binary relations between the individuals and individual names representing single individuals in the domain. The $\mathcal{SROIQ}$-roles and $\mathcal{SROIQ}$-concepts can be built upon atomic concepts (A), atomic roles R_A, top concept $\top$, bottom concept $\bot$, named individuals (o_i), simple roles (S) and universal role U by using different constructors shown in Table 1. A DL-based ontology $O = (\mathcal{T}, \mathcal{A})$ consists of a finite set $\mathcal{T}$ (TBox) of terminology axioms and a set $\mathcal{A}$ (ABox) of individual axioms. A TBox may include concept inclusion axioms, role inclusion axioms, transitive role axioms, disjoint role axioms, reflexive role axioms, irreflexive role axioms, symmetric role axioms and asymmetric role axioms. An ABox may include concept assertions, role assertions, negated role assertions, inequality assertions and equality assertions (see Table 1). Note that $\mathcal{SROIQ}(\mathcal{D})$ is obtained through extending $\mathcal{SROIQ}$ with datatypes, which are entities that refer to sets of data values such as strings and numbers. So far, the roles mentioned in Table 1 are abstract roles whose domain and range are concepts. If the range of a role is datatype literal, this role is called datatype role [36]. The concepts, abstract roles and datatype roles in DLs correspond to the classes, object properties and data properties in OWL respectively.

2.1.2. Description logic semantics

The semantics of $\mathcal{SROIQ}$ is defined by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ interprets concepts, roles and