An Initial Investigation into Querying an Untrustworthy and Inconsistent Web

Yuanbo Guo and Jeff Heflin

Dept. of Computer Science and Engineering, Lehigh University, Bethlehem, PA18015, USA
{yug2, heflin}@cse.lehigh.edu

Abstract. The Semantic Web is bound to be untrustworthy and inconsistent. In this paper, we present an initial approach for obtaining useful information in such an environment. In particular, we replace the question of whether an assertion is entailed by the entire Semantic Web with two other queries. The first asks if a specific statement is entailed given an identification of the trusted documents. The second asks for the document sets that entail a specific statement. We propose a mechanism for efficiently computing and representing the contexts of the statements and managing inconsistency. This system could be seen as a component in an overall trust system.

1 The Problem

Since the Semantic Web is intended to mirror the World Wide Web, it will be produced by numerous information providers with different levels of credibility, and will be used by information consumers who have different opinions on who or what is trustworthy. Researchers have investigated methods for developing trust systems in such environments. Some work on building trust networks between individuals (e.g. [3, 4]) while others focus on determining whether to trust Web content depending on its sources (e.g. [5]). In this work, we take a different approach which deals with the untrustworthy Semantic Web from a perspective of extensional queries. We investigate how to build a Semantic Web search engine that can tell the user what sources support each answer to such a query. In addition, we aim at a system that could efficiently answering queries once the user has decided what sources they trust and allow them to create hypothesis about trusted documents and explore the consequences of their choices.

We will assume a document collection D consisting of N OWL Lite [2] documents, labeled as d_1 to d_n. We also assume that this collection can be harvested from the Internet at a rate such that the information maintained by a webcrawler is current enough to be of value. Note, our focus on a centralized search-engine approach is based on its success in the contemporary Web and on the fact that much research needs to be done before distributed queries can reach a comparable response time. Finally, we will assume that users are primarily interested in extensional queries, and will focus on queries about the instances of a class. We use a:C to denote an assertion that individual a is an instance of class C.

We introduce two definitions before formally defining our problem. First, we say a set of documents D entails a statement φ iff φ is entailed by the union of the imports

closure of every document in D. As such, it is possible that a pair of documents might entail something that is not entailed by either document alone. Second, a set $D_{sub} \subseteq D$ is a minimal consistent subset of D that entails φ iff D_{sub} is consistent, D_{sub} entails φ, and there is no subset of it that entails φ. Note, for a given φ, there may be multiple such sets. Based on these, we propose two kinds of queries to be answered by the system:

- Q1: Given a trusted subset D_{sub}, is D_{sub} consistent and does it entail an assertion a:C?
- Q2: What are the minimal consistent subsets of D that entail an assertion a:C?

As can be seen, we have taken inconsistency into account in the queries. In classical logic, everything can be deduced from an inconsistent knowledge base. This is not desirable for Semantic Web applications and it is crucial to be able to identify inconsistent document sets. Much work has been done in the logic community to study paraconsistent logics that allow reasoning with inconsistent information, e.g., [6, 7]. Unlike this work, we suggest that inconsistency can be managed, not by changing the underlying logic, but by changing the kind of queries we pose to the Semantic Web.

It needs to be noted that although this work is not about a trust system in a strict sense, it is possible to integrate our system with existing trust systems. For instance, another system could determine the trusted set for Q1. More importantly, we take this work as a first step to build a new trust system, wherein query answering as described above serves as an instrument for deciding trust on resources. This might be a system that automatically determines trust taking account of the results of Q2, or a system that assists the user in deciding what to trust by revealing the query results to him.

2 An Initial Approach

A naïve way to answer the above queries is as follows. To answer Q1, we first combine the documents in D_{sub} by loading them into a single knowledge base. Then we check the knowledge base. If it is found inconsistent, the answer to Q1 is no. Otherwise, we query about a:C on the knowledge base. The result is then the answer to Q1. Such a query can be executed using a description logic reasoner that supports realization. To answer Q2, we repeat Q1 against each applicable subset of D. We enumerate the subsets of D in increasing order of their sizes. In order to ensure that only minimal consistent subsets are returned, we keep track of those subsets that either answer yes to Q1 or are inconsistent so that we could skip all the supersets of them later on. This works because OWL is monotonic. The answer to Q2 will then be the document sets which have answered positively to Q1. However, this naïve approach has several drawbacks. First, it is incapable of reusing the results of the expensive reasoning from the preceding queries. For instance, if a Q1 query is repeated, we have to carry out the same process all over again. Answering Q2 is similar in this aspect. Second, the scalability of this approach is a problem especially for answering Q2. Again, the complexity cannot be amortized over multiple queries.

Our approach, generally speaking, aims at improving the scalability and efficiency through the reuse the results of document processing, especially reasoning. This is realized by adding to the document processing a new functionality of figuring out and recording the "context" of each encountered statement. Here we define the "context" of a statement as a minimal consistent document set that entails the statement.

We employ an assumption-based truth maintenance system (ATMS) [1] to streamline the management of such contexts. In an ordinary ATMS, each node is associated with a proposition and a justification is a record of logical inference made between those propositions. In our approach, we use ATMS in an unconventional way. We use the ATMS nodes to represent two types of objects. We use an assumption node to represent a single document from D. We call the node a document node. And we use a derived node to represent a set of documents, in other words, the combination of these documents. We call the node a combination node. Following the notation in [1], we use Γ_d to denote a document node representing document d, and γ_v a combination node representing document set v. A justification $\Gamma_{d1},\ldots,\Gamma_{dn}=>\gamma_v$ is then interpreted as: the conjunction of the statements entailed by documents $d_1,\ldots,d_n$ implies the statements entailed by the document set v. Moreover, a justification $\Gamma_{d1},\ldots,\Gamma_{dn}=>\perp$ conveys the information that document set $\{d_1,\ldots,d_n\}$ is inconsistent. It can be interpreted in a similar way when the antecedents of the justification contain combination nodes. Fig. 1 is an example ATMS after four documents are added, among which $\{d_1, d_2\}$ and $\{d_1, d_3\}$ are inconsistent.

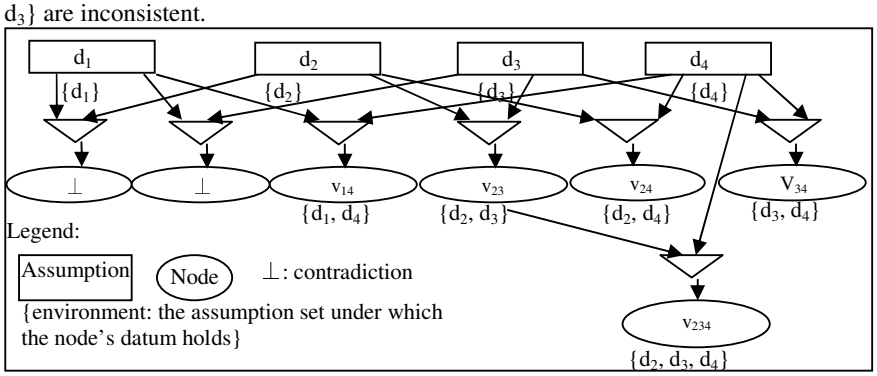

Fig. 1. An example ATMS network

There are several reasons for us to associate an ATMS node with a document or a document set as opposed to a statement. First is the scalability consideration. The scale of data makes it impossible to represent each statement individually and to provide a specified justification for it. Second, we assume that documents are all or nothing, i.e., we trust either the whole content of a document or none of it. A more pragmatic reason is that since our description logic reasoners are black boxes, we cannot easily determine exact justifications at the level of a statement. We instead must determine them at the document level. As a result, an ATMS node in our system essentially points to a set of statements and it serves as the media of the context of those statements: an environment of such a node is just a minimal consistent document set which entails the set of statements associated with the node.

Now what we need to do is to store the statements together with their contexts. To make our system more scalable, we do not store the deductive closure of the knowledge base. We observe that once subsumption has been computed, a simple semantic network is sufficient for answering queries about the instances of classes. Therefore, we only store the subsumption relations which are not redundant (for example, $C1 \subseteq C3$ is redundant given $C1 \subseteq C2$ and $C2 \subseteq C3$) and the most specific classes of each instance.

However, to answer the queries presented in Section 1, we also need context information. As a result, what is stored can be seen as a semantic network whose links are "annotated" by the contexts, as depicted by Fig. 2. As we will show soon, this allows us to replace the expensive description logic reasoning with a much simpler semantic network inference-like procedure during query answering. In this way, we find a balance between doing some precomputation at loading time in order to save query time while controlling storage requirements.

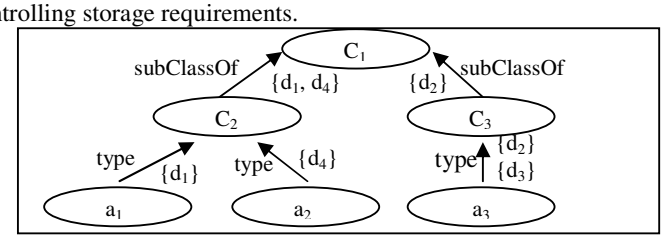

Fig. 2. An "Annotated" Semantic Network

In order to scale to a large number of classes and instances, we implement the "annotated" semantic network using a relational database. We use two kinds of tables. The first is a set of class instance tables, where there is one for each class. Each tuple of the table contains an individual of the class and a pointer to an ATMS node, whose environment can entail that the corresponding individual is an instance of the class. We call that node a supporting node. Since the same concept assertion may hold in different document sets, an individual may have multiple supporting nodes with respect to a specific class. The second kind of table is a class taxonomy table, which records the class subsumption. Each tuple in the table consists of a superclass, its subclass, and a supporting node for the subsumption relation. Again, a subsumption may be supported by multiple nodes. Table 1 and Table 2 show what these tables look like for the semantic network displayed in Fig. 2.

Table 1. Class Instance Tables

Class C_2

Individual	Supporting Node
a_1	d_1
a_2	d_4

Class C_3

Individual	Supporting Node
a_3	d_2
a_3	d_3

Table 2. Class Taxonomy Table

SuperClass	SubClass	Supporting Node
C_1	C_2	v_{14}[1]
C_1	C_3	d_2

The ATMS and the tables are updated as documents are loaded and processed. Due to space constraints, we omit the operational details. The basic idea is that when a document is newly added, we insert it to the ATMS, apply inference on it, and store the entailed statements to the tables. Then we combine it with each consistent subset of the preceding documents, reflect the combination on the ATMS, reason about it, and store the entailed statements by the combination. In addition, once a new set of inconsistent documents is detected during the processing, we record it in the ATMS.

Given this preprocessing, we can make the query answering more lightweight by reducing them to simple operations involving multiple table lookups. First consider Q1 with respect to individual *a*, class *C*, and a set of documents *set*. If *set* is inconsistent according to the ATMS, we return no to the query. Otherwise, we search for *a* in *C*'s

[1] combination node of d_1 and d_4

table. For each found tuple of *a*, we look up the ATMS for the environment of its supporting node. If the environment is a subset of *set*, we answer yes to the query. If we could not directly find a matching tuple in the instance table, we will resort to the class taxonomy table. We search for the subclasses of *C* in *set*, and repeat the test with those subclasses. Q2 can be answered in a similar way, except that when two elements, one from the taxonomy table and the other from an instance table, are used simultaneously to derive an answer, we add the union of the environments of their support nodes to the result. Moreover, we guarantee that what are finally returned are only those minimal environments, i.e., document sets.

Compared to the naïve algorithm, our approach reduces query time at the cost of doing extra work when loading documents. However, considering the significant improvement on query efficiency, we could argue that the complexity of the document processing in advance could be amortized over a large number of queries. Furthermore, the complexity can be alleviated in the case when a document set is identified inconsistent at some time and a significant number of combinations involving that set are avoided later on. It is similar in the case when some documents import others.

3 Future Work

For future work, we will look into ways to further improve the scalability of our approach, especially to reduce the average cost in the preprocessing. One plan is to devise a mechanism that discovers in advance if nothing new can be entailed by a combination of documents and thus allows us to omit the combination. Also we intend to transfer the current approach into a distributed one. In addition, we will extend the system to support other kinds of queries such as queries about role assertion, and more complex queries comprised of multiple atoms. Finally, as we mentioned, our long term goal is to build a trust system on top of this work.

References

1. Kleer, J. de. An assumption-based TMS. Artificial Intelligence, 28(2), 1986.
2. Dean, M. and Schreiber, G ed. OWL Reference. http://www.w3.org/TR/owl-ref/
3. Golbeck, J., Parsia, B., and Hendler, J. Trust networks on the Semantic Web. In Proc. of Cooperative Intelligent Agents. 2003.
4. Richardson, M., Agrawal, R., and Domingos, P. Trust Management for the Semantic Web. In Proc. of ISWC2003.
5. Gil, Y. and Ratnakar V. Trusting Information Sources One Citizen at a Time. In Proc. of ISWC2002.
6. Fitting, M.C. Logic Programming on a Topological Bilattice. Fundamenta Informaticae, 11(1988).
7. Blair, H.A. and Subrahmanian, V.S. Paraconsistent Logic Programming. Theoretical Computer Science, 68(1989).