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 statement is entailed if a set of documents is trusted. The second asks for the document sets that entail a specific statement. We propose a mechanism which leverages research on assumption-based truth maintenance systems to efficiently compute and represent the contexts of the statements and manage inconsistency. For these queries, our approach provides significant improvement over the naïve solution to the problem.

1 Introduction

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. Some researchers have investigated methods for computing who a user should trust in such environments. In this work, we take a different approach: we investigate how to build a Semantic Web search engine that can tell the user what sources support each answer to a query so that the user could decide if they trust those sources. In addition, we aim at a system capable of efficiently answering queries once the user has decided what sources they trust and when they change their mind.

We will assume a document collection D consisting of N OWL Lite [3] 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 denote by a:C an assertion that individual a is an instance of class C.

Before we can formally define our problem, we must introduce two definitions. First, a set of documents D entails a statement φ iff φ is entailed by the union of the imports closure [4] 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, and 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 this, we propose two kinds of queries to be answered by the system:

- Q1: Given a trusted subset D_{sub} (of size M)[1], 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?

We also take into account inconsistency in the queries. In classical logic, everything can be deduced from an inconsistent knowledge base. However, in many Semantic Web applications, this is not desirable and it is crucial to be able to identify inconsistent document sets in order to avoid inappropriate use of the semantic data. By these queries, 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.

The rest of the paper is organized as follows. Section 2 looks at the naïve approach to answer the above two queries. Section 3 describes in detail an improved approach. Section 4 discusses related work. Section 5 concludes.

2 A Naïve Approach

1) Answering Q1

To answer Q1, we first combine the documents in D_{sub}. This involves loading them into a single knowledge base. Then if the knowledge base is found inconsistent, the answer to Q1 is false; 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 such as Racer [6].

2) Answering Q2

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 during the test.

Next we analyze the complexity of this approach. To facilitate the analysis, we assume that the average size of the documents in D is S_D, and so is the average size of the documents in $D_{sub.}$

1) Answering Q1

It is known that reasoning on a language like OWL Lite, which maps to SHIQ(D+), is expensive with worst case N_{exp}Time complexity. Therefore, we could expect that the dominant factor of the complexity of answering Q1 is the time spent on reasoning including consistency check and instance query. We denote it by $T_{inf}(M*S_D)$. For simplification, in the subsequent discussion, we will not make distinction between different sorts of reasoning processes, instead, we generally refer to their time complexity as

[1] This set might be explicitly specified by the user or be determined by some certification authority that the user has specified.

$T_{inf}(s)$ wherein s is the size of the knowledge base measured by the total size of the documents loaded into it.

2) Answering Q2

Suppose k is the total number of subsets that have been tested with Q1, then the best case occurs when every document in D either entails the target assertion or is inconsistent. In that case, k equals to N and the time complexity is $N*T_{Q1}(S_D)$, wherein $T_{Q1}(s)$ stands for the time complexity of answering Q1 on a document set of size s. On the contrary, the worst case happens when none of the subsets could be skipped by the strategy and we are forced to do the query on all of them. In that case, k is as large as 2^N-1, and the time complexity is $O(2^N)* T_{Q1}(N*S_D)$.

This approach has several drawbacks. First, it is incapable of reusing the results of expensive inference 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.

3 An Improved Approach

3.1 Assumption-Based Truth Maintenance System

Our approach builds on the concept of assumption-based truth maintenance system (ATMS) [1]. Like the conventional justification-based truth maintenance system (JTMS) [2], ATMS makes a clear division between the problem solver and the TMS, as shown in Fig. 1. The TMS functions as a cache for all the inference made by the problem solver. Thus inferences, once made, need not be repeated, and contradictions, once discovered, are avoided in the future. Unlike JTMS which is based on manipulating justifications, ATMS is, in addition, based on manipulating assumption sets. In an ATMS, every datum is labeled with the sets of assumptions, a.k.a. environments, under which they hold. These assumption sets are computed by the ATMS from the problem solver supplied justifications. Consequently, ATMS makes it possible to directly refer to a context, defined as a given environment plus all the data derivable from it. Thus context switching, which is very expensive in JTMS, is free in ATMS. Therefore, ATMS can work more efficiently than JTMS for problem solving by exploring multiple contexts simultaneously.

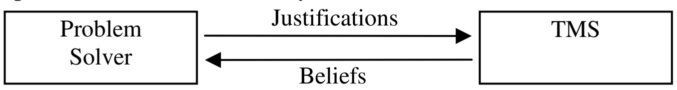

Fig. 1. ATMS Components

3.2 Document Preprocessing

Generally speaking, our approach aims at improving the scalability and efficiency by reusing 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: a "context" of a statement is a minimal consistent document set that entails the statement. This definition is sufficient because OWL is monotonic: if a statement is entailed by a set of documents, it is also entailed by any superset of that set; likewise, if a document set is inconsistent, each of its supersets will be inconsistent too.

We adopt ATMS 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 $\Gamma_{d1},\ldots,\Gamma_{dn}\Rightarrow\gamma_v$ 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 represented by v. Moreover, a justification $\Gamma_{d1},\ldots,\Gamma_{dn}\Rightarrow\perp$ 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. 2 is an example ATMS for four documents, among which {D1, D2}, {D1, D3} and {D3, D4} are inconsistent. We will introduce the algorithm for constructing such ATMS at the end of the section.

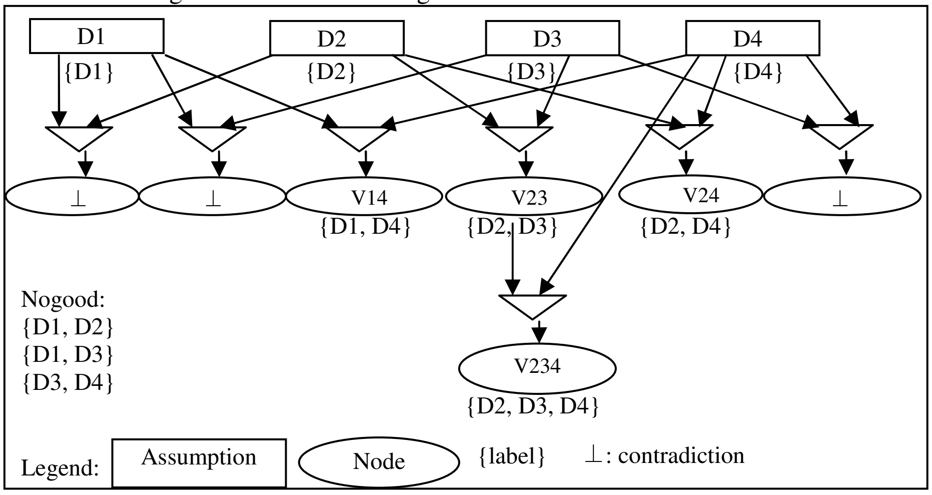

Fig. 2. An example ATMS network

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. One other minor 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: the environment of such a node is just a minimal consistent document set which entails the 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, $C1 \sqsubseteq C3$ is redundant given $C1 \sqsubseteq C2$ and $C2 \sqsubseteq C3$) 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. 3. As we will show in next section, 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.

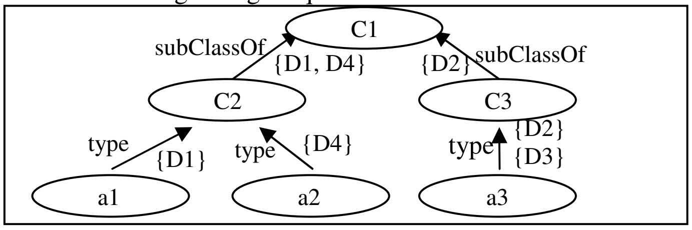

Fig. 3. An "Annotated" Semantic Network

Also for scalability, we store the "annotated" semantic network in 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 supporting node. By supporting nodes, we mean those nodes whose environment can entail that the corresponding individual is an instance of the class. Recall that a node's environment is a set of documents. 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. 3.

Table 1. Class Instance Tables

Class C2

Individual	Supporting Node
a1	D1
a2	D4

Class C3

Individual	Supporting Node
a3	D2
a3	D3

Table 2. Class Taxonomy Table (*combination node of D1 and D4)

SuperClass	SubClass	Supporting Node
C1	C2	V14*
C1	C3	D2

Now we give the algorithm for processing a document (PROCESS-DOC). Due to space constraints, we intersperse the description within the pseudo code. The basic idea is, when a document is newly added, we apply inference on it and store the statements entailed by it. Then we combine it with each of applicable subsets of the preceding documents and store the entailed statements by the combination.

```
procedure PROCESS-DOC($D_k$)
{Assuming documents $D_1,\ldots,D_{k-1}$ have already been processed}
begin
  LOAD($D_k$);  /*load the document into a knowledge base*/
  ADD-DOC($D_k$);
  if $D_k$ is consistent then
    for each non-empty subset s of {$D_1,\ldots,D_{k-1}$} do[2]
      if ATMS: CHECK-NOGOOD(s)=false[3] then COMBINE-DOCS(s, $D_k$);
end
procedure ADD-DOC($D_k$)
begin
  ATMS: ADD-ASSUMPTION($D_k$); /*add an assumption node for $D_k$*/
  DO-INFERENCE($D_k$);  /*apply DL inference on $D_k$*/
  if $D_k$ is inconsistent then ATMS:NOTIFY-JUSTIFICATION("$\Gamma_{Dk}$=> $\perp$");
  else STORE-STATEMENTS($D_k$, $\Gamma_{Dk}$);
end
procedure COMBINE-DOCS(set, $D_k$)
begin
  set_node := the representative node of set;
  $V_{new}$ := LOAD(set U {$D_k$}); /*load $D_k$ and all documents in set
          into a knowledge base, $V_{new}$ being the combination*/
  DO-INFERENCE($V_{new}$); /*apply DL inference on $V_{new}$*/
  if $V_{new}$ is inconsistent then
    ATMS: NOTIFY-JUSTIFICATION("set_node, $\Gamma_{Dk}$  => $\perp$");
  else begin
    ATMS: ADD-NODE($V_{new}$); /*add a new node representing $V_{new}$*/
    ATMS: NOTIFY-JUSTIFICATION("set_node, $\Gamma_{Dk}$ => $\gamma_{vnew}$");
    STORE-STATEMENTS($V_{new}$, $\gamma_{vnew}$);
  end
end
procedure STORE-STATEMENTS(d, node)
{node is the ATMS node representing document set d;}
begin
  for each non-redundant concept axiom C1$\subseteq$C2 in d do
    ADD-TO-TAXONOMY-TABLE[4](C1$\subseteq$C2, node);
  for each concept assertion a:C in d wherein C is the most
  specific class of a do
    ADD-TO-INSTANCE-TABLE[5](a:C, node);
end
```

[2] We ignore it for brevity, but owl:imports can be taken into account for document combination. For instance, if one document imports another, we can skip the combination of both of them.

[3] One important functionality of ATMS is to record the assumption sets that have caused contradictions in a so-called nogood list. For instance, if we notify the ATMS that D_k is inconsistent, it will record {D_k} as a nogood environment.

[4,5] Both procedures guarantee that a statement will ultimately be stored only with the nodes representing its contexts, i.e., the minimal consistent document sets that entail the statement.

3.3 Query Answering

Based on the above preprocessing, we can make the query answering more lightweight by reducing them to simple operations involving multiple table lookups. The algorithms are listed below. TEST-INSTANCE answers Q1 with individual *a*, class *C*, and a set of documents *set*. If *set* is inconsistent, we return false to the query (Line 4). Otherwise, we search for *a* in *C*'s table (Line 5, TEST-INSTANCE1). If we find a tuple of *a* such as its supporting node has an environment which is subset of *set*, we answer yes to the query (Line 24). If we could not directly find a matching tuple in the instance table, we will resort to the class taxonomy table (Lines 6-16). We search for the subclasses of *C* in *set*, and repeat the test with those subclasses (Line 12).

```
 1 procedure TEST-INSTANCE(a, C, set) return true or false
 2 begin
 3   {a: an individual; C: a class; set: a subset of D;}
 4   if ATMS: CHECK-NOGOOD(set)=true then return false;
 5   if TEST-INSTANCE1(a, C, set)=true then return true;
 6   else begin
 7     search the class taxonomy table for C;
 8     for each found tuple t do begin
 9       n := t.SupportingNode;
10       env := ATMS: GET-ENVIRONMENT(n);
11       if env ⊆ set then
12         if TEST-INSTANCE(a, t.SubClass, set)=true then
13          return true;
14     end
15     return false;
16   end
17 end
18 procedure TEST-INSTANCE1(a, C, set) return true or false
19 begin
20   search the instance table of C for a;
21   for each found tuple t do begin
22     n := t.SupportingNode;
23     env := GET-ENVIRONMENT(n);
24     if env ⊆ set then return true;
25   end
26   return false;
27 end
```

QUERY-INSTANCE answers Q2, also by consulting the information in the tables. But unlike TEST-INSTANCE, when two elements, one from the taxonomy table and the other from an instance table, are used simultaneously to derive an answer, the algorithm adds the union of the environments of their support nodes to the result (Line 20). In addition, it guarantees that what are finally returned are only those minimal environments, i.e., document sets. This is covered by INSERT-CONTEXT and INSERT-CONTEXTS in Lines 4, 9 and 20.

```
1 procedure QUERY-INSTANCE(a, C) return a document set
2 {a: an individual; C: a class; results := {};}
3 begin
4   INSERT-CONTEXTS(results, QUERY-INSTANCE1(a, C, {});
  search the class taxonomy table for C;
  for each found tuple t do begin
    n := t.SupportingNode;
    env := GET-ENVIRONMENT(n);
    INSERT-CONTEXTS(results,QUERY-INSTANCE(a,t.SubClass,env));
  end
  return results;
end
procedure QUERY-INSTANCE1(a, C, set) return a document set
{a: an individual; C: a class; results := {};}
begin
  search the instance table of C for a;
  for each found tuple t do begin
    n := t.SupportingNode;
    env := GET-ENVIRONMENT(n);
    INSERT-CONTEXT(results, set U env);
  end
  return results;
end
```

3.4 Complexity Analysis

Now we analyze the computational complexity of our approach. As with the naïve approach, we focus on the most significant operations.

1) Answering Q1

Our approach has reduced query answering to table searching and eliminated the need of doing inference. How many database operations are required depends on how many document sets the statements considered in the process have as their contexts. In the best case, the number is at constant level. In the worst case, however, the number is $O(2^N)$, e.g., when the statement in the query is entailed by the maximum number of subsets of D such that no set contains another. Nevertheless, we could expect that in a real application, most of the statements will only have a handful of, if not one, document sets in D as their contexts. Therefore, the time complexity will be very close to the best case.

2) Answering Q2

As shown in QUERY-INSTANCE, answering Q2 has been realized in similar algorithm to that of Q1, except that the algorithm has to examine all possible contexts of a statement since no candidate is specified as in Q1. But this does not increase the order of complexity. In other words, we have achieved a complexity of Q2 similar to 1). This is a significant improvement compared to the naïve approach.

3) Document preprocessing

Our approach reduces query time by doing extra work when loading documents. There are three major kinds of work: ATMS related operations, database operations, and inference. The most significant task is inference. ADD-DOC does inference on documents while COMBINE-DOCS does inference on document sets. Therefore roughly, the time complexity of both procedures are $T_{inf}(N*S_D)$. Since we try to combine a newly added document with every subset constituted by its preceding document, there are potentially $O(2^N)$ such subsets, which means COMBINE-DOCS has to be invoked for $O(2^N)$ times in the worst case. This results in a worst case complexity of $O(2^N) *T_{inf}(N*S_D)$.

However, this 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. The example in Fig. 2 demonstrates this. It is similar in the case when some documents import others. In addition, considering the 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, since queries are significantly faster here than in the naïve approach.

4 Related Work

Trust systems for the Semantic Web as in [7, 8, 9, 10] are developed to compute whether to trust a Semantic Web resource depending on certain factors such as its source. Clearly our work is not about such kind of trust system. Our system deals with the untrustworthy Semantic Web from a perspective of extensional queries and leaves the determination of who to trust in the hands of the user. However, it is possible to integrate our system with other trust systems. For example, another system could determine the trusted set for Q1, or the results of our Q2 could be used as input into a system trying to determine some form of community based trust.

Sesame is another Semantic Web system that makes use of a truth maintenance system. In particular, it uses a simplified JTMS to track all the deductive dependencies between statements and to determine which other statements have to be removed as a consequence of a single deletion [5]. Since we are essentially dealing with multiple contexts, the ATMS is much more suitable for us. In addition, their work tries to find out the dependency between statements while ours deals with the justification on statements at document level.

Finally, much work has been done in the logic community to study paraconsistent logics that allow reasoning with inconsistent information. Examples of such logical systems include bilattice-based logics [11, 12] and annotated logics [13, 14, 15]. As noted at the beginning of the paper, our work does not aim at developing the underlying logic to handle inconsistency. Instead, we proposed to manage inconsistency by changing the kind of queries we pose to the Semantic Web.

5 Conclusions and Future Work

In this paper, we considered the issue of how to obtain useful information on the inherently untrustworthy and inconsistent Semantic Web. We proposed an approach to build a system that could answer two kinds of queries. One asks if a statement is entailed by a trusted set of documents and that set is consistent. The other asks for the minimal consistent document sets that entail a specific statement as a way to help the user to decide if they trust those sources. We employ an assumption-based truth maintenance system (ATMS) to efficiently represent document sets as the contexts of the statements entailed by them. Also we leverage the mechanism of ATMS for inconsistency management. Based on that, we introduced a mechanism which preprocesses the documents and caches the complex inference together with statement context information, and answer the queries based on these. We showed how our approach greatly improves the efficiency with respect to the proposed queries. Another characteristic of our approach is that it seamlessly integrates the task of query answering and inconsistency management in one framework. In this paper, we have concentrated on the queries about concept assertions. However, the approach presented here can be easily extended to support role assertions, for example.

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 of our plans 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, possibly based on the work on distributed ATMS like [16].

References

1. Kleer, J. de. An assumption-based TMS. Artificial Intelligence, 28(2), 1986.
2. Doyle, J. A truth maintenance system. Artificial Intelligence 12(1979).
3. Bechhofer, S. et al. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/
4. Patel-Schneider, P.F. ed. OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/
5. Broekstra, J. and Kampman, A. Inferencing and Truth Maintenance in RDF Schema: exploring a naive practical approach. In Workshop on Practical and Scalable Semantic Systems (PSSS). 2003.
6. Haarslev, V. and Moller, R. Racer: A Core Inference Engine for the Semantic Web. In Workshop on Evaluation on Ontology-based Tools, ISWC2003.
7. Golbeck, J., Parsia, B., and Hendler, J.Trust networks on the semantic web. In Proc. of Cooperative Intelligent Agents. 2003.
8. Klyne, G. Framework for Security and Trust Standards. In SWAD-Europe. 2002.
9. Richardson, M., Agrawal, R., and Domingos, P. Trust Management for the Semantic Web. In Proc. of ISWC2003.
10. Gil, Y. and Ratnakar V. Trusting Information Sources One Citizen at a Time. In Proc. of ISWC2002.
11. Ginsberg, M.L. Multivalued logics: A uniform approach to inference in artificial intelligence. Computer Intelligence, 4(1988).
12. Fitting, M.C. Logic Programming on a Topological Bilattice. Fundamenta Informaticae, 11(1988).
13. Subrahmanian, V.S. On the Semantics of Quantitative Logic Programs . In IEEE Symposium on Logic Programming. 1987.
14. Blair, H.A. and Subrahmanian, V.S. Paraconsistent Logic Programming. Theoretical Computer Science, 68(1989).
15. Kifer, M. and Lozinskii, E.L. A logic for reasoning with inconsistency. Journal of Automated Reasoning, 9(2), 1992.
16. Malheiro, B., Jennings, N., and Oliveira, E. Belief Revision in Multi-Agent Systems. In Proc. of the 11th European Conference on Artificial Intelligence (ECAI'94). 1994.