An Interactive Diagnosis and Repair of OWL Ontology

Lu Lu, Nandan Parameswaran
Department of Computer Science and Engineering
University of New South Wales
Sydney , Australia, 2031
{lulu, paramesh}@cse.unsw.edu.au

Pradeep Kumar Ray
School of Information Systems Technology and Management
University of New South Wales
Sydney , Australia, 2031
p.ray@unsw.edu.au

Abstract

Ontology diagnosis is an important phase in the process of knowledge base development and management. In the various applications of semantic computing, ontology is not static, but keeps changing over time due to changes in the local environments. A change is typically effected by adding new axioms or modifying parts of existing axioms in the ontology. Such arbitrary changes might result in conflicts in the knowledge base and how to diagnose and repair the conflict in the knowledge base is an interesting but hard problem. In this paper, we propose an interactive strategy for maintaining knowledge base consistency by identifying the unsatisfiable concepts in the knowledge base, and eliminating them using suggestions obtained from the knowledge engineer (user).

1. Introduction

Ontology diagnosis is a crucial phase in knowledge development and management. In the various applications of semantic computing, one is faced with the situation where ontology changes dynamically, potentially introducing conflicts (unsatisfiable concepts) into the knowledge base. Detecting and eliminating the unsatisfiable concepts is one of the central problems to ontology management. Although some OWL tools have focused on various aspects of ontology engineering, the support for diagnosis and repair defects in OWL ontologies has been fairly weak. Common defects include inconsistent ontologies and unsatisfiable concepts[5]. These defects have logical contradictions (errors), so they can be detected by DL (description logic) reasoner. However, DL reasoners do not explain the reason for the error nor do they offer any support for resolving the error.

The main problem we discuss in this paper is how one can analyze the axioms in an ontology for any logical errors and arrive at strategies for revising the unsatisfiable concepts such that the ontologies may become consistent. The basic idea involves computing MUPSes minimal unsatisfiable preserving sub-TBox) from a given inconsistent ontology and generating queries from the MUPSes where the query requires a YES or NO answer from the user, Once an answer is obtained from the user, a new axiom is generated, and any other axioms which are in conflict with this axiom are either modified or removed. Compared to the traditional method where users need to read and analyze all the axioms in the MUPS and revise the conflicting concepts manually, our strategy is more convenient for users in repairing the ontology.

Using the notions of frequent axioms[11] and root and derived classes[5], we select an unsatisfiable concept for repair and rearrange axioms of its MUPS. While our technique is somewhat similar to the tableaux tracing algorithm[6], we observe that the output of the tableaux tracing algorithm are not suitable for direct machine processing.

The main contribution of this paper are as follows.
(1) Given a MUPS that is untraceable, we analyze the reason as to why it is untraceable, and present a method to transform the untraceable MUPS to the traceable one.
(2) We propose a simple strategy for acquiring additional knowledge from the user by generating simple queries from the MUPSes, use that knowledge to update the knowledge base, and make it consistent.

The rest of the paper is organized as follows. In section 2, we present the related work. In section 3, we present an intuitive example showing how our algorithm works. Section 4 presents the details of our interactive strategy for ontology repair. In section 5, we conclude the paper with some final remarks.

0-7695-2997-6/07 $25.00 © 2007 IEEE
DOI 10.1109/ICSC.2007.35

2. Related Work

Works related to ontology diagnosis can be separated into two classes: belief revision and diagnosis in reasoning system. The core idea in belief revision is how to change a given knowledge base, such that the overall change to the knowledge bases itself is kept minimal [7][3].

The other area is diagnosis in reasoning system. In [10], the author introduces the notion of conflict set and minimal hitting set, and proposes an algorithm to compute the minimal hitting set from the conflict set. In [11], the computational complexity of this algorithm has been improved by proposing a heuristic where the notion of MUPS (minimal unsatisfiable preserving sub-TBox) was defined and used.

In [11], techniques have been proposed for detecting the unsatisfiable concepts from an ontology. However, it only lists the frequent axioms of all the MUPSs. The analysis of these axioms for revising the unsatisfiable concepts is still done manually.

The core idea of their [11] heuristic algorithm is to calculate the most frequent axioms in the MUPSs. The problem with the heuristic algorithm is that the set comprised of the most frequent axioms is not guaranteed to be the minimal hitting set.

Aditya [5] introduces two concepts, called root class and derived class where he assigns higher priority to root classes for revision. However, if there are more than one root classes, it is left to the user to choose an appropriate root class to begin with.

One problem with the MUPS is that if the axioms in the MUPS are presented in a random sequence, it is hard (for user) to understand. In [6], Kalyanpur,et al. propose a tableaux tracing algorithm to arrange the axioms of the MUPS in a sequence that is easy for users to understand. Different from [11], who focus on the most frequent axioms, they focus on the unsatisfiable concepts, and give the definition of "root class" and "derived class"[5]. Because we also use the concept "root class" and "derived class", here we borrow their definition in [5].

Some other related work is in explaining inferences for description logic systems. [8] provides explanations for the CLASSIC KR system. [12] proposes non-standard reasoning algorithms (for ALC TBoxes) and use DICE terminology to demonstrate their algorithm. Some popular ontology editors such as Protégé[9] and OilEd [2] provide little or no explanation for diagnosis and revise the onologies.

3. An intuitive Example

We begin with a few definitions.

Definition 1 (MUPS).A TBox $T' \subseteq T$ is a minimal unsatisfiability preserving sub-TBox (MUPS) for A (A is a concept) in T, if A is unsatisfiable in T', and A is satisfiable in every sub-TBox $T'' \subseteq T'$. [11]

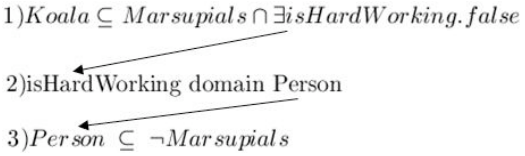

Figure 1. rearranged MUPS of "Koala"

Definition 2 (Root Class) Root Class is an unsatisfiable atomic class in which a clash or contradiction found in the class definition (axioms) does not depend on the unsatisfiability of another atomic class in the ontology.[5]

Definition 3 (Derived Class) Derived Class is an unsatisfiable atomic class in which a clash or contradiction found in a class definition either directly (via explicit assertions) or indirectly (via inferences) depends on the unsatisfiability of another atomic class.[5]

Consider the MUPS from koala.owl[1] in Example 1.

Example 1: (MUPS1 is for koala, MUPS2 is for quokka.)
MUPS1:
1)$Koala \subseteq \exists isHardWorking.false$;
2) isHardWorking domain Person ;
3)$Person \subseteq \neg Marsupials$;
4)$Koala \subseteq Marsupials.$

MUPS2:
1) $Quokka \subseteq Marsupials$;
2)$Quokka \subseteq \exists isHardWorking.true$;
3) isHardWorking domain Person ;
4) $Person \subseteq \neg Marsupials.$

In this example, the most frequent axioms of the TBox are axiom 2 and axiom 3 of MUPS 1. Selecting the frequent axioms for repair does not necessarily make the ontology conflict free. For example, if we only consider the most frequent axioms 2 and 3 in Example 1, we find that these two axioms are conflict free. However, when we consider them long with the unsatisfiable concepts ("Koala") we will find the cause of the error - that is, the domain of "isHardWorking" is not Koala. In this example because "Koala" has a derived class, and "Quokka" do not have, we select "Koala" for repair firstly.

Fig 1 shows the rearranged MUPS for "Koala" as a tree. Notice that we have combined axioms 1 and 4 in the original MUPS to derive the root of the tree. The root of the tree gives rise to two branches, one corresponding to "Marsupials" and the other to "$\exists isHardWorking.false$". Each of these concepts will be referred to as the first concept of

[1] http://www.mindswap.org/2005/debugging/ontologies/koala.owl

its branch. (Thus, "$\exists isHardWorking.false$" is the first concept of the second branch.) Using axiom 2 and axiom 3, we now construct two more nodes for the second branch with the following rational: Tracing back from the bottom, axiom 3 means "Person" is not the subset of "Marsupials". Then we can infer from axiom 2 and 3 that "isHardWorking domain is not Marsupials". We thus find that "not Marsupials" in this branch is in conflict with "Marsupials" (the first concept of the first branch). Then we can generate a query, is "not Marsupials" the domain of "isHardWorking"? If the answer is yes, we will generate a new axiom, "isHardWorking domain ¬ Marsupials", and use it to update the knowledge base. Thus, none of the subsets of Marsupials should have the role isHardWorking, then the axioms $Koala \subseteq \exists isHardWorking.false$ and $Quokka \subseteq \exists isHardWorking.true$ should be removed. If the answer is no, it means axiom 2 or axiom 3 in MUPS 1 is false. We now generate a query, Is person the domain of isHardWorking? If the user says YES, it means axiom 3 is an error one, then we infer a new axiom "$Person \subseteq Marsupials$", delete all other axioms that conflict with this new axiom. (We present the complete details in Section 4.5)

In this paper, we assume users would give the correct answer, so the answer of " is "not Marsupials" the domain of "isHardWorking"?" is YES. Note that "isHardWorking domain ¬ Marsupials" is not an axiom in the original knowledge base. In this example, if we had only considered the axioms in the MUPS, we might have removed the axiom $Koala \subseteq \exists isHardWorking.false$ but it would have been of no help in solving the problem of Quokka in MUPS 2.

4. The Algorithm

We now present an algorithm for generating the queries to assist in the repair of a given ontology starting from a given MUPS.

4.1. Preliminaries

We will use DL SHOIQ[4] to describe concepts.

SHOIQ-roles contain inclusion, hierarchy, transitivity and the inverse relation on roles. SHOIQ-concepts contain concept name C, $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$, ($\leqslant$ n R.C), $(\geqslant nR.C)$, where R is a SHOIQ-role.

Based on these definitions, we can identify three types of relations: (1) relation between concepts(we call it T_1). (for example, $C_1 \subseteq C_2$).(2)domain/range relation between role and concept (T_2). (3) relation between roles(T_3). In the rest of paper, we use the following notations: let C_r denote the concept that contains a role, C_{nr} the concept that does not contain a role, C_{rr} the concept in the range of a role, C_{dr} the concept in the domain of a role. Further, we let C_r denote a root class, C_d is a derived class, and **UC** an unsatisfiable concept. If the left hand side of an axiom is a concept, for example C, we call this axiom the axiom of C.

In [1], the author gives mentions about "ABox containing a clash" (page 86). We can identify two types of clashes: 1) {C(x),$\neg C(x)$} $\subseteq$ ABox for some individual name x and some concept name C. In this paper, we call concept C as "**conflict concept**". In the rest of the paper, we use "**CC**" stands for conflict concept.
2) { $(\leqslant nR)(x)$, $(> mR)(x)$ } $\subseteq$ ABox, for nonnegative integers $n < m$.

In this paper, we discussed the first case. The method to solve the second case is somewhat simple. We let the tableaux algorithm return the domain of R (suppose C_{dr}) and the number n, and then generate a query: is relation of the role R and the n in the domain C_{dr}? There are four options: 1)$\leqslant n$ 2) $> m$ 3) $\leqslant x$ 4)$> y$. The user can either select options 1 and 2, or can specify the values of x and y such that the options 3 and 4 are satisfied.

4.2. Unsatisfiable Concept Selection

We need to first select a MUPS for revision. While in [11], the idea of "pinpointing" (which involves choosing a MUPS in which the most frequent axiom of all MUPS occurs) is used, and in [5] the notion of root class is used, we in our approach have combined these two ideas to select a MUPS of a root class in which a most frequent axiom occurs. If there is more than one root class that contains the most frequent axiom, we select the root class that has more derived classes. If the MUPS for an unsatisfiable concept C1 contains all the axioms of the MUPS for the unsatisfiable concept C2, C2 is the "root class" of C1, and C1 is the "derived class" of C2.

4.3. Construction of the MUPS Tree

We begin the construction of the tree by letting the axiom of the unsatisfiable concept(UC) be the root node(N_r) of the MUPS tree . The other nodes of the MUPS tree are formed using the other axioms in the MUPS. Figure 2 gives the rules of the MUPS tree construction. Rule-1 is to construct the root. In Rule-1 we do not consider the case UC $\subseteq$ concepts, and UC $\equiv$ concepts. In section 4.4, we will discuss this case. Rule-2 and Rule-3 are used to construct other nodes. In Rule-3, $\neg C \equiv concepts$, $concepts \equiv C$ and $concepts \equiv \neg C$ are equivalent to $C \equiv concepts$.

Figure 3 shows the MUPS tree for Example 2(from University.owl[2].)

Example 2(MUPS for $CS\_Department$)
1) $CS\_Department \subseteq \exists affiliatedWith.CS\_Library$
2) $EE\_Department \subseteq \exists affiliatedWith.EE\_Library$

[2]http://www.mindswap.org/2005/debugging/ontologies/University.owl

Rules of the MUPS Tree Creation	
Rule-1,	If $UC \subseteq$ concepts, and concepts $\subseteq \neg UC$, then $UC \subseteq$ concepts $\cap \neg$ concepts
Rule-2,	If R is the role of the concept C, and the axiom AX is the domain/range relation of R, then AX is the child of the concept C
Rule-3,	If the axiom AX has the form $C \subseteq$ concepts, concepts $\subseteq \neg C$, or $C \equiv$ concepts, then AX is the child of the concept C

Figure 2. the Rules of the MUPS Creation

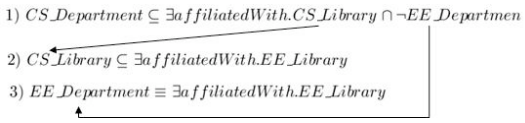

Figure 3. MUPS tree for "CS-Department"

3) $EE\_Department \subseteq \neg CS\_Department$
4) $CS\_Library \subseteq \exists affiliatedWith.EE\_Library$
5) $Transitive(affiliatedWith)$

The following is the algorithm of the MUPS tree construction. We associate an axiom with every node of the tree. Let S_{C_i} is the set of concepts, associated with node i. If the current node is T_1(we describe T_1 in section 4.1), get all the concepts on the right hand side of the axiom, and store them into S_{C_i}. If the current node is T_2, store the concept of the axiom into S_{C_i}. For example, 1) in Figure 3 is T_1, thus S_{C_i} =$\{\exists affiliatedWith.CS\_Library, \neg EE\_Department\ \}$. In the section 4.1, we describe 3 types of the axioms. The algorithm only use T_1 and T_2 to create the MUPS tree. T_3 is stored in a role property set(we call it **RPS**). In Example 2, 5)"Transitive(affiliatedWith)" is stored in RPS. The detail of how to use RPS and MUPS tree is described in section 4.5. The output of the algorithm is the MUPS tree, RPS and unused axioms set. Because of the limited space, we do not use pseudocode to present the algorithm. To make the presentation simple, we use "go to" instead of "while".

Algorithm (MUPS Tree Construction)

1), Use Rule-1 to generate the root node(N_r), current node = N_r, and store the T_3 axioms into RPS, T_1, T_2 axioms into unused axioms set

2), Based on current node, generate S_{C_i}.

3), For each the concept in S_{C_i}, if any one is C_r, but not the conflict concept, use Rule-2, remove the concept from S_{C_i}. If the concept has a child node, remove current node from unused axioms set and set the child node as the current node and go to 2).

4), For each the concept in S_{C_i}, if any one is C_{rr}, but not the conflict concept, use Rule-3, remove the concept from S_{C_i}. If the concept has a child node, remove current node from unused axioms set and set the child node as the current node and go to 2).

5), For each the concepts in S_{C_i}, if any one is C_{nr}, but not the conflict concept, use Rule-3, remove the concept from S_{C_i}. If the concept has a child node, remove current node from unused axioms set and set the child node as the current node and go to 2).

6), If current node is not the root node, and S_{C_i} is empty, then set the father node as the current node and go to 3); if S_{C_i} is not empty,then go to 3); else end the algorithm.

Figure 4. MUPS for "CS-Department"

In the algorithm, 3),4) and 5) means if there are C_r, C_{rr} and C_{nr}, we handle C_r first, then C_{rr}, C_{nr}. C_r owns highest priority.

We use Rule-1 to transform the 3) in Example 2 to "$CS\_Department \subseteq \neg EE\_Department$" and combine it with 1) in Example 2. 1) in Figure 3 is the root. "$\exists affiliateWith.CS\_Library$" is the first concept of a branch, and concept($CS\_Library$) is the range of '$\exists affiliateWith.CS\_Library$", because of Rule-3, 2) in Figure 3 is the child of this concept. "$\neg EE\_Department$" of 1) is the first concept of another branch, because of Rule-3, 3) in Figure 3 is the child of this concept.

The idea used in Figure 3 to construct the MUPS tree for Example2 is similar to the tableaux tracing algorithm[6], but there are two differences. Figure 4 shows the output of the tableaux tracing algorithm for Example 2.

(1) The tableaux tracing algorithm calculates the trace of MUPS by extending the expansion rule of the tableaux algorithm. One problem of the tableaux tracing algorithm is some traces of the derived class are not necessary to be calculated. When the root class is solved, its derived class will be solved or will become a root class anyway. Our algorithm only need to calculate the traces of some of the root classes(See Section 4.5).

(2)The aim of the tableaux tracing algorithm is to present the users an understandable MUPS where the axioms have originally occurred in the TBox. However, in our case, we rewrite and combine the axioms in order to construct the root and other nodes of the tree. In a way, this makes ontology repair easier

	Rule to Solve the Untraceable MUPS
Rule-1,	IF 1. A MUPS M is untraceable; 2. M contains at least one pair axioms, the form: $A \equiv D$ and $A \subseteq C$, or $A \equiv D$ and $C \subseteq \neg A$; Then transform $A \equiv D$ in the pairs to $A \subseteq D$

Figure 5. Rule to Solve the Untraceable MUPS

4.4. Traceable and Untraceable MUPS

While constructing a MUPS tree, we may expect that for each axiom in the MUPS there will be a node in the tree labeled by the axiom, so that the unused axioms set is empty. If this is the case, we call such a MUPS as traceable. (The MUPS in Figure 1 is traceable.) However, a given MUPS may not be traceable (untraceable). For example, in Example 3 (from the file badfood.owl[3]) below, if we use our tree construction algorithm, for the axioms 3, 4 and 5, there are no tree nodes for which these axioms are the labels, the unused axiom set is not empty, and thus this MUPS is untraceable. This can happen in situations (during MUPS generation process) when any formulas generated by the absorption algorithm[1] and used by the tableaux algorithm are not added to MUPS.

Example 3 (MUPS for "VeganFood")
1)$VeganFood \equiv \neg(Dairy - products - and - eggs \cup Meat) \cap Food$;
2) $Dairy - products - and - eggs \equiv Eggs \cup Dairy$;
3) $LactoVegetarianFood \equiv Food \cap \neg(Eggs \cup Meat)$;
4) $OvoVegetarianFood \subseteq \neg LactoVegetarianFood$;
5)$OvoVegetarianFood \equiv \neg(Dairy \cup Meat) \cap Food$.

We find that the absorption algorithm in [1] (Page 334-335) will generate some new formula under this condition: If the $A \equiv D$, $A \subseteq C$, and there is no axiom of D in the TBox. Then $A \subseteq C$ will be transformed to $D \subseteq C$, and is added to T-Box, but not to the MUPS, thus causing the MUPS untraceable. One way to make this MUPS traceable is to add the new formula $D \subseteq C$ to the MUPS, thus making it traceable. In the experiment, we found that when a MUPS contains axioms of the form $A \equiv D$ and $A \subseteq C$, the reason for the unsatisfiability of the MUPS is because the definition for A is too strict: that is $A \equiv D$ and $A \subseteq C$ at the same time. Since $A \equiv D$ is equivalent to $A \subseteq D$ and $D \subseteq A$. If we drop $D \subseteq A$, the MUPS will disappear. Thus we propose Rule-1(see Figure 5) that not only transforms the untraceable MUPS to traceable MUPS, but also solve the clash sometimes.

In the experiment, we find that in some situations, the traceable MUPS also contains the axioms which form is $A \equiv B$ and $A \subseteq C$, or $A \equiv B$ and $C \subseteq \neg A$. When we transform the operator $\equiv$ to $\subseteq$, the unsatisfiable concept becomes satisfiable. The reason is the same as the untraceable MUPS. To solve this problem, we change the condition 1 "A MUPS M is untraceable;" to "a new axiom which is not contained in the original TBox, is generated by the absorption algorithm;".

In Rule 1 of section 4.3, we do not consider the case UC $\subseteq$ concepts, and UC $\equiv$ concepts. Now use the rule in this section, $\equiv$ will be transformed to $\subseteq$, the two axioms will be combined as one.

4.5. *AND* Branch

In this section(and in Section 4.6 where we consider the *OR* branch) we propose strategies for revising MUPSs so that logical errors in knowledge bases can be eliminated.

If $C\_r_1 ... C\_r_n$ are the root classes of (a derived class) C_1, when $C\_r_1 ... C\_r_n$ become satisfiable, C_1 also becomes satisfiable, or else it becomes a root class. We thus only consider MUPSs of root classes for revision. In a MUPS tree, there are two kinds of branches: *AND* branch and *OR* branch. In this section, we will discuss the case the MUPS only contains the *AND* branch.

Definition 4(*AND* branch). In a MUPS tree, the sequence of nodes appearing from the root to a conflict concept at the leaf, where none of the axioms contain a disjunction, is called an *AND* branch. For example, in Figure 3, axiom 1 is the root node, axiom 2 is the leaf node, and the sequence <1 , 2> is an *AND* branch, (Sequence <1, 3> is another *AND* branch. Similarly, in Figure 1, < 1, 2 , 3> and $< 1 >$ are *AND* branches). We will denote the length of the branch from node N_i to node N_j by $L(N_i, N_j)$.

As described in the section 4.1, the conflict concept will appear as a pair(CC, $\neg CC$) in the MUPS. Because of the way our algorithm constructs the tree, CC must appear at the leaf node of the tree. (We will denote such nodes by N_{CC} or $N_{\neg CC}$. Also, we assume there is only one pair of conflict concepts. If there are more, we can invoke this algorithm more than once.)

We observe that there are 3 ways to solve the conflict concept: (1) remove some concept in a node; (2) negate some concept in a node; (3) weaken the root axiom. Before we present the algorithm, we use Figure 6 to illustrate the idea.

The trees shown Fig 6a and Fig 6c are MUPS trees. Figure 6b is not a MUPS tree but a graph showing the dependency structure between the concepts in the axioms. (Further explanation is given below.) The nodes in Figure 6a and 6c are associated with axioms. The label of the broken line is the answer to the query generated from the tree. Based on this query and the answer, we can generate a new axiom, and use it to update the knowledge base. The rules

[3]http://www.mindswap.org/2005/debugging/ontologies/bad-food.owl

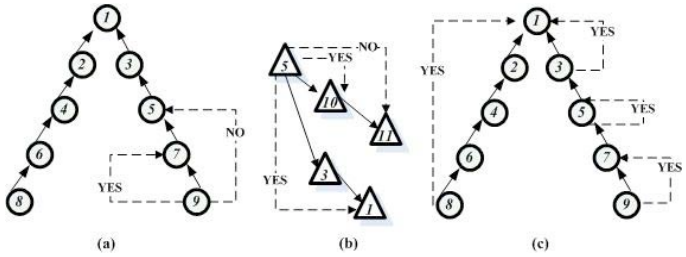

Figure 6. Example of AND branch

to generate the query and the new axiom are in Figure 7 (Rule-1, Rule-2 and Rule-3). If the new axiom, for example, "C_5 is the domain of $\neg R_1$"(it means each subset of C_5 should not have role R1 as its domain), then we will use this axiom to update the knowledge base(it is shown in Figure 6b). We use the dependency graph Fig 6b to perform the updation of the knowledge base. Each node in this graph is labeled with a concept, and the edge in the graph represents the $\subseteq$ relation. In this case in Fig 6b above the graph can be interpreted to denote the following relationship: $C_{10} \subseteq C_5$, $C_{11} \subseteq \exists R_1 C_{12} \cap C_{10}$, $C_3 \subseteq C_5$, $C_1 \subseteq \exists R_1 C_{13} \cap C_3$. Before we remove the concept "$\exists R_1 C_{12}$" , we must generate a query "is C_{11} the subset of C_5". In this example, since the answer is no, we need to find the error axiom of concepts C_5,C_{10} and C_{11}. (We use binary search technique to locate the error axiom.) Since the answer to the query (which is generated by node 5 and 10) is YES, we can infer that the error axiom is at node 11; that is, the axiom $C_{11} \subseteq \exists R_1 C_{12} \cap C_{10}$ can be modified as $C_{11} \subseteq \exists R_1 C_{12} \cap \neg C_{10}$ by negating the concept C_{10}.

After the update of the knowledge base, we check the knowledge base to see if the conflict concept still persists. Conflict concepts can still persist if the update we performed above happened in some disjoint part of the knowledge base, thus leaving the MUPS tree unaffected. If this was the case, we use Function 1 below to find the error axiom where we have used binary search techniques again to locate the error axiom. In this example (Fig 6a), we negate the parent concept of node 7, derive a new formula and use it to update the knowledge base (as we did in Figure 6b).

Finally, if all the axioms turn out to be correct (according to the user) as, for example, shown in Figure 6c, our last option is to weaken the root axiom. Consider the following example: 1) $PhD \subseteq student$; 2) $PhD \subseteq employee$; 3) $employee \subseteq \neg student$. If all these three axioms are correct (that is, none of the conflicts could be removed), then we select one concept out of the set {students , employee}. For example, we can select student and modify the knowledge base as: $student \subseteq common\_student \cup special\_student$, $PhD \subseteq special\_students$, $employee \subseteq \neg common\_students$.

The final algorithm is given below where we have used

Rule-1 If (1) $C_1 \subseteq R.C_2, ...C_{n-1} \subseteq R.C_n$,
(2)$C_1 \subseteq R.C_2$ is the root class, or $C \subseteq R_1.C_1$,
or C_1 is the domain/range of R_1.
(3) $R_1 \neq R$, R is transitive,
then the query is, is R the role of C_1 and C_n?
If the answer is yes, then the new axiom is $C_1 \subseteq R.C_n$.
If the answer is no, then the new axiom is $C_1 \subseteq \neg R.C_n$.

Rule-2 If C_1 is the domain/range of R_1, and the answer of Rule-1 is yes.
then the query, is C_1 the domain/range of R_1?
If the answer is yes, then the new axiom is "C_1 the domain/range of R_1".
If the answer is no, then the new axiom is "C_1 the domain/range of $\neg R_1$".

Rule-3 If C_1 is the domain/range of R_1, $C_1 \subseteq C_2$, $...C_{n-1} \subseteq C_n$,
then the query, is C_n the domain/range of R_1?
If the answer is yes, then the new axiom is "C_n the domain/range of R_1".
If the answer is no, then the new axiom is "C_n the domain/range of $\neg R_1$".

Figure 7. Rule for AND branch

rules in Fig 7 to construct the new axioms. To make the algorithm easier to understand, we divide it into four modules, the algorithm as a framework, Function 1 to (binary) search for the error axiom, Function 2 to update the knowledge base, and finally Function 3 to weaken the axiom.

Rules in Figure 7 are used to derive new axioms by generating a query and asking the user to answer YES or NO.(Notice that we have not shown the universal and existential quantifiers in the rules for brevity.)

We now present the Algorithm for query generation and knowledge base update. (To keep the algorithm more intuitive, we have used "go to" instead of "while".)

Algorithm (Query Generation and Knowledge base update)

let N_c is the current node(where the algorithm generates the query). N_{nc} is the parent node of N_c, N_r is the root node, N_s is the start node, N_e is the end node. At the beginning, $N_c = N_s = N_{cc}$, $R.C_n$ is the conflict concept, $N_e = N_r$. If the answer is yes, $N_s = N_c = N_{nc}$, and $R.C_n$ is the concept that has a child node. If the answer is no, $N_e = N_c$, N_s does not change and $N_c = N_s$. N_m is the middle node between N_s and N_e, m =$[\frac{L(N_s,N_e)}{2}]$.

1) Select the longest *AND* branch. If all the *AND* branches contain the same number of the axiom, then select the branch for which the first concept contains role. Suppose the leaf node in this branch is N_{CC}.

2) If the current node is not equal to the root node

2.1) Use Rules in Figure 7 to generate the query.

2.2) Use Function 2 to update the knowledge base.

2.3) If the MUPS tree contains N_{CC} and $N_{\neg CC}$ and the an-

swer of Rules in Figures 7 is yes, then go to 2); else if the answer of Rules in Figures 7 is no, use Function 1 to find the error axiom and go to 3).
3) If the MUPS tree does not contain N_{CC} or $N_{\neg CC}$, then end the algorithm.
4) If the MUPS tree contains N_{CC} and $N_{\neg CC}$, these nodes are used in the algorithm, then use Function 3 to weaken the root axiom , end the algorithm, else go to 2).

The Function 1 (Error Axiom Search)
1) If N_s equals N_e, then negate the concept that has child node in N_s, end this function.
2) If N_m equals N_e, then set N_e as the only input of Rule-1, use Rule-1 to generate the query, .
2.1) Use Function 2 to update the knowledge base. If the answer of the query is yes, negate the conflict concept ,or the concept that has child node in N_s, end this function. (Note that N_s and N_e are updated after each query)
3) If N_m does not equal N_e, then set the nodes between N_m and N_s as the input of Rule-1, use Rule-1 generate the query.
3.1) Use Function 2 to update the knowledge base.
3.2) If the MUPS tree does not contain N_{CC} or $N_{\neg CC}$, end this function; else update the N_m, go to 1).

The Function 2 (Knowledge Base Update)
1) If the new axiom(AX_1) is T_1(as we describe in section 4.1), then search all subset of the concept(C) which on the left hand side of AX_1, and store the concepts in the set(S_c). If the new axiom(AX_1) is T_2, and the concept(C) is the domain of the role, then search all subset of the C of AX_1, and store the concepts in the set(S_c). If the new axiom(AX_1) is T_2, and the concept(C) is the range of the role, then search all the axioms that contains the role in the knowledge base, and store the axioms in the set(S_c).
2) For each the concept(C_i) in S_c, if there is an axiom of C_i conflict with AX_1, generate the query, is C_i the subset of C? For each axiom in S_c, if any one conflict with AX_1, suppose C_{rr} is the concept in the range of the role, generate the query, is C_{rr} the range of the role?
2.1) If the answer is yes, and (AX_1) is T_1 , then negate the concept of the axiom which is conflict with AX_1, end this function. If the answer is yes, and (AX_1) is T_2 , then remove the concept of the axiom which is conflict with AX_1, end this function.
2.2) If the answer is no, then find the error axiom and fix it (it is similar with Function 1), end this function.

The Function 3 (Weaken the Axiom)
1) C_1 is the first concept of one *AND* branch, C_2 is the first concept of the other *AND* branch. Suppose we select C_1.
2) add a new axiom, $C_1 \subseteq special\_C_1 \cup common\_C_1$
3) transform the axiom of UC to $UC \subseteq special\_C_1 \cap C_2$
4) use $common\_C_1$ to replace other C_1 in the knowledge base.

We will now summarize the functioning of the algorithm by showing how to eliminate the conflict in Figure 3. In Figure 3, there are two *AND* branches, and the length of these branches is the same. Because $\exists affiliatedWith.CS\_Library$ contains role, we select this branch first. Use Rule 1, we will generate a query, is affiliatedWith the role of $CS\_Department$ and $EE\_Library$? The answer is no. When we use Function 1 to find the error axiom, because N_m equals N_e, we generate one more query, is affiliatedWith the role of $CS\_Department$ and $CS\_Library$? The answer is Yes. The N_s is $CS\_Library \subseteq \exists affiliatedWith.EE\_Library$, then we negate the conflict concept, the new axiom is $CS\_Library \subseteq \neg\exists affiliatedWith.EE\_Library$. Now the conflict is eliminated.

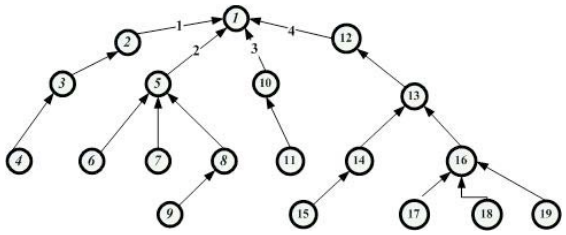

Figure 8. Example of OR branch

In Figure 1, the longest branch is $< 1, 2, 3 >$, we use Rule 3 to generate the query, is $\neg Marsupials$ the domain of isHardWorking? The answer is yes. When we will use Function 2 to update the knowledge base, we generate two query: (1) is Koala the subset of Marsupials? (2) is Quokka the subset of Marsupials? The answer is yes. Then we remove axiom 1 in MUPS1 and axiom 2 in MUPS2. This time, we not only solve the unsatisfiable concept "Koala", but also "Quokka".

4.6. *OR* Branch

In this section, we will discuss *OR* branch in the MUPS.

Definition 4(*OR* branch). In a MUPS tree, if any of the axioms contains a disjunction, the tree is said to contain an *OR* branch.

In the section 4.5, if there are n pairs of conflict concepts, the algorithm only handles one pair each time. In this section, because the axioms contain the disjunction, we separate the pairs of conflict concepts into different groups, and the algorithm only handles one group each time. For example, $C_1 \subseteq (C_2 \cup C_3) \cap \neg C_2 \cap \neg C_3 \cap (C_4 \cup C_5) \cap \neg C_4 \cap \neg C_5$, there are 4 pairs conflict concepts: (C_2, $\neg C_2$),(C_3, $\neg C_3$),(C_4, $\neg C_4$),(C_5, $\neg C_5$). They are separated into 2 groups, one is (C_2, $\neg C_2$)and(C_3, $\neg C_3$), the other is (C_4, $\neg C_4$) and (C_5, $\neg C_5$).

In this section, the MUPS tree contains *AND* branches

and *OR* branches. Figure 8 is an example of the MUPS tree that contains *OR* branches. N_1 is the root, there are 4 branches in this MUPS tree. Branch 1 and branch 3 are *AND* branches, branch 2 and branch 4 are *OR* branches.

As we have noted before, in the MUPS tree construction, N_{CC} occurs at the leaf node. In the *OR* branch, the node contains the disjunction is call *OR* node(N_{or}). We can regard the branch that from N_{CC} to N_{or} as an *AND* branch. The branch from one N_{or} to its upper N_{or} is also an *AND* branch. Thus we can view *OR* branches case as a set of *AND* branches. Although there are more than one pairs of the conflict concepts, if any one conflict concept is solved, the whole problem is solved.

The core idea of the algorithm of *OR* branch case involves searching the *AND* branch first. In figure 8, branch 1 and branch 3 are *AND* branch, we use the algorithm in section 4.5 for these branches. If any conflict concept in these branches disappears, end the algorithm. If all the axioms in the *AND* branches are correct, then search the *OR* branches. Suppose we search branch 2 first. We start from node 9(the lowest leaf of branch 2), the branch between node 9 and node 5 is an *AND* branch. If the axioms in this branch are correct, then search the branch(node 6 and 5), branch (node 7 and 5). If all the branches under node 5 are correct, then search the branch (node 5 and 1). It is similar to search the branch 4. If all the axioms in the MUPS are correct by users, we user the function 4 in section 4.5 to weaken the root axiom.

5 Conclusion and Future work

In this paper, we propose an interactive strategy towards repairing these unsatisfiable concepts. The core idea is automatically generating the query based on the unsatisfiable concepts and the MUPSs. Consider the limited pages, we briefly describe the algorithm to automatically generate the query and how to handle people's reply in the *AND* branch and the *OR* branch cases. Currently we are working on improved diagnostic and revision techniques and evaluating their performances in situation management applications.

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press.

[2] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: a reason-able ontology editor for the semantic web. Proceedings of KI2001, Joint German/Austrian Conference on Artificial Intelligence,, 2001.

[3] P. Gardenfors. Belief revision: An introduction. *Cambridge Tracts in Theoretical Computer Science*, 1992.

[4] I. Horrocks and U. Sattler. A tableaux decision procedure for shoiq. Proc. of the 19th Int. Joint Conf. on Artificial Intelligence, 2005.

[5] A. Kalyanpur. *Debugging and repair of OWL ontologies*. PhD thesis, 2006.

[6] A. Kalyanpur, B. Parsia, B. C. Grau, and E. Sirin. Tableaux tracing in shoin. Technical report, Dept. of Computer Science, University of Maryland, 2005.

[7] G. P. Markinson and D. Alchourron. the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 1985.

[8] D. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, 1996.

[9] M. N. Noy, S. Sintek, M. Decker, R. F. Crubezy, and M. Musen. Creating semantic web contents with protege-2000. IEEE Intelligent Systems,, 2001.

[10] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 1987.

[11] S. S., H. Z., C. R., and V. H. F. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 2007.

[12] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. Proc. of the 17th Int. Joint Conf. on Artificial Intelligence, 2003.