Ontology Inconsistency Handling: Ranking and Rewriting Axioms

Sik Chun (Joey) Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos

Department of Computing Science
University of Aberdeen, AB24 3UE, UK
{slam, jpan, sleeman, wvasconc}@csd.abdn.ac.uk

Abstract. Ontology reasoners are able to detect inconsistencies in ontologies; however, there is relatively limited support for resolving the problems. Existing approaches either pinpoint axioms which are responsible for inconsistencies or calculate maximal consistent/coherent sub-ontologies of inconsistent/incoherent ontologies. These approaches simply remove problematic axioms from the ontology, the support for correcting the axioms is still limited. In this paper, we identify two typical scenarios of inconsistent/incoherent ontologies and extend existing approaches by proposing some methods for ranking and rewriting axioms in ontologies, so as to provide users with guidelines on how to achieve a consistent/coherent ontology. We implement these methods in a prototype system, aiming at enabling non-expert users to resolve inconsistencies.

1 Introduction

Ontologies [15] play an important role in the Semantic Web [1]. The advent of the Web makes an increasing number of ontologies widely available for reuse. After adding new axioms into an existing ontology, the user may find that the revised ontology becomes inconsistent. While reasoners are able to detect inconsistencies in ontologies, there is relatively limited support for resolving the problems. It should be noted that there are two kinds of inconsistencies in ontologies. An ontology is inconsistent iff it has no model; an ontology is incoherent iff it contains an unsatisfiable named concept. For now we focus on incoherent ontologies which are represented in the description logic $\mathcal{ALC}$ [14]. Our approach, however, can be trivially extended to more expressive DLs and inconsistent ontologies.

There are two existing approaches to deal with inconsistent ontologies: (1) to pinpoint the so called Minimal Unsatisfiability Preserving Sub-ontologies (MUPSs) [13], [7], which are sets of axioms responsible for an unsatisfiable concept, and (2) to calculate Maximally Concept-Satisfiable Sub-ontologies (MCSSs) [10], in which satisfiable sub-ontologies are obtained by removing just enough axioms to eliminate all errors. These two approaches are very similar because they identify subsets of an ontology's axioms which are possible sources of the problems. In the case of MUPSs, the sets of problematic axioms are identified directly; in the case of MCSSs it is the axioms excluded from an MCSS which are problematic. The later approach is believed to have better scalability, as a tableau-like algorithm is used to identify MCSSs; the former develops pinpointing algorithms as additional functionality to an external DL reasoner.

Both approaches identify sets of problematic axioms, and then leave it up to the user to modify the errors or provide the user with a set of maximal coherent ontologies. However, there is no further support for selecting which problematic axioms to remove or modify, and in the case of modification, no support is provided to assist with the rewriting of the axioms.

What is needed are: (1) methods to rank potentially problematic axioms to give the user guidance on which ones should be removed or modified, and (2) methods to rewrite the problematic axioms. We now illustrate two typical scenarios of inconsistent/incoherent ontologies to illustrate the need for these methods.

1. *inconsistencies as errors* – People may find it difficult to understand the logical meaning of the underlying description logic, and potential inferences in ontologies; hence people may fail to formulate axioms which are logically correct, or may specify contradictory statements. For example,
   (a) $\mathsf{Lion} \sqsubseteq \exists\, \mathsf{eats}\, (\mathsf{Sheep} \sqcap\ \mathsf{Cow})$
   (b) $\mathsf{Sheep} \sqsubseteq \neg\, \mathsf{Cow}$
   (c) $\mathsf{Sheep} \sqsubseteq \mathsf{Animal}$
   (d) $\mathsf{Cow} \sqsubseteq \mathsf{Animal}$
   Either axiom (a) or (b) should be removed from the ontology, however, we believe that sibling concepts are usually disjoint with each other, therefore, axiom (a) is suggested to be removed or rewritten. This case shows the need for ranking axioms in an ontology in order to assist the user in choosing which axioms to remove or rewrite.
2. *inconsistencies as knowledge overwrite* – a typical example is the bird ontology in which 'birds can fly' ($\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$). One may extend this ontology by stating 'penguins are birds which cannot fly' ($\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$, $\mathsf{Penguin} \sqsubseteq \neg\, \mathsf{CanFly}$). This would lead to an incoherent ontology, because it implies penguins can both fly and not fly. Though removing one of the axioms can resolve the problem, this approach is not reliable, because the removal may cause significant information loss in the ontology, hence it would be better to rewrite one of the axioms. Therefore, it is necessary to give the user guidance on how to rewrite axioms in order to resolve the contradiction. For instance, the default axiom $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ can be rewritten such that all birds but penguin can fly, i.e. $\mathsf{Bird} \sqcap \neg\, \mathsf{Penguin} \sqsubseteq \mathsf{CanFly}$.

In this paper, we first introduce the existing approach to finding Maximal Concept-Satisfiable Sub-ontologies (MCSSs), in which axioms causing the inconsistency are excluded. We go a step further by introducing our adapted and novel techniques to indicate the likely correctness of axioms in an ontology. To do this, we propose knowledge heuristics based on: (i) ontology structure, and (ii) patterns of usage of the ontology, which will be described in Section 3. Hence we enable users to select the MCSS which is most likely to be correct. The next step is to correct the axioms which are excluded from MCSSs, rather than to directly remove them from the ontology. We provide change operations for the users to rewrite the excluded axioms: the concepts in such axioms can be generalised or specialised (see Section 4). The user will be presented with a number of rewriting options. Each possible rewrite means the loss of some entailments from the original ontology. We want to ensure that the modified ontology can be as "close" as possible to the original ontology (based on the Principle of Minimal Change [8]). We

achieve this by sorting the rewriting suggestions returned to the user, to give preference to the most specific generalisations (or the most general specialisations).

We will show, using examples, how our method is applicable in ontologies with change annotations, and also applications which reuse/integrate ontologies, as well as ontology evolution. This work provides an integrated framework for both ranking and rewriting problematic axioms. The framework has been implemented (see Figure 1A), in the tool ReTAX++ [9]. With our approach, we enable even users who are not experts in ontology engineering to resolve inconsistencies.

This paper is organised as follows: Section 2 describes the necessary formal definitions of an ontology etc. In Section 3, we present methods for selecting a sub-ontology by ranking the axioms. Strategies for rewriting axioms are presented in Section 4. We present the related work and conclusion in Sections 5 and 6.

2 Preliminaries

2.1 Ontology Model

We will describe our proposed framework in terms of the underlying OWL ontology model, which is represented in the description logic $\mathcal{ALC}$ [14]. Let N_C and N_R be a set of concept names and relation names respectively; an ontology $\mathcal{O}$ consists of a set of *terminology* axioms $\mathcal{T}$ (TBox) and assertional axioms $\mathcal{A}$ (ABox). An axiom in $\mathcal{T}$ is of the form $C \sqsubseteq D$ where C and D are arbitrary concepts; an axiom in $\mathcal{A}$ is of the form $a : C$ where C is a concept and a is an individual name. A concept is defined by the following syntactic rules, where CN is a concept name, R is a relation, C_1 and C_2 are concept expressions:

$$\top \mid \bot \mid \mathsf{CN} \mid \neg C_1 \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C_1 \mid \forall R.C_1$$

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set of individuals and the interpretation function $(\cdot^{\mathcal{I}})$ maps each concept name $\mathsf{CN} \in N_C$ to a set $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each relation name $\mathsf{RN} \in N_R$ to a binary relation $\mathsf{RN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Given a $\mathcal{T}$ and a concept name A, $\mathcal{T}$ is *A-satisfiable* iff there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$. $\mathcal{T}$ is *concept-satisfiable* iff it is A-satisfiable for every concept name A occurring in $\mathcal{T}$. $\mathcal{T}$ is *unfoldable* iff the left-hand side of every $\alpha \in \mathcal{T}$ contains a concept name A, there are no other α's with A on the left-hand side, and the right-hand side of α contains no direct or indirect references to A [10].

2.2 Finding Maximally Concept-Satisfiable Sub-ontologies

In this subsection, we revisit the existing approach to finding the maximally concept-satisfiable sub-terminologies of an unfoldable terminology. Meyer et al. [10] provide a specialised tableau-like algorithm for finding the maximally A-satisfiable subsets (A-MSSs) of an unfoldable terminology $\mathcal{T}$ represented in $\mathcal{ALC}$, for any concept name A occurring in $\mathcal{T}$. We then describe how to extend their work in the subsequent sections.

Example

Table 1 shows an example ontology $\mathcal{O}$ containing a set of terminology axioms $\mathcal{T}$, that

we will use for the rest of this paper. The ontology contains two unsatisfiable concepts, PhDStudent and MScStudent. We will use this example to explain how to achieve a coherent ontology.

Table 1. An ontology example $\mathcal{O}$

α_1. PGCourse $\sqsubseteq \neg$ UGCourse	α_7. PhDStudent $\sqsubseteq$ PGStudent
α_2. UGCourse $\sqsubseteq$ Course	α_8. MScStudent $\sqsubseteq$ PGStudent
α_3. PGCourse $\sqsubseteq$ Course	α_9. MPhilStudent $\sqsubseteq \forall$ take. Course
α_4. PGStudent $\sqsubseteq$ Student	α_{10}. MScStudent $\sqsubseteq \forall$ take.(PGCourse $\sqcap$ Exam)
α_5. UGStudent $\sqsubseteq$ Student	α_{11}. PhDStudent $\sqsubseteq \forall$ take.PGCourse
α_6. MPhilStudent $\sqsubseteq$ PGStudent	α_{12}. Student $\sqsubseteq \exists$ register.Dept $\sqcap \exists$ take.UGCourse
	α_{13}. PGStudent $\sqsubseteq \exists$ take.($\neg$ Exam)

Maximally A-satisfiable subsets (A-MSSs)
A subset $\mathcal{T}'$ of $\mathcal{T}$ is an A-MSS of $\mathcal{T}$ iff it is A-satisfiable, and every $\mathcal{T}''$ such that $\mathcal{T}' \subset \mathcal{T}'' \subseteq \mathcal{T}$ is A-unsatisfiable. Intuitively, a set of A-MSS is obtained by excluding axioms involved in the clash. We abbreviate the set of all A-MSSs of A in $\mathcal{T}$ by $mss(A)$. From Table 1, the A-MSSs of the unsatisfiable concepts in our example are:
mss(PhDStudent) = $\{\mathcal{T} \setminus \{\alpha_1\}, \mathcal{T} \setminus \{\alpha_4\}, \mathcal{T} \setminus \{\alpha_7\}, \mathcal{T} \setminus \{\alpha_{11}\}, \mathcal{T} \setminus \{\alpha_{12}\}\}$
mss(MScStudent) = $\{\mathcal{T} \setminus \{\alpha_8\}, \mathcal{T} \setminus \{\alpha_{10}\}, \mathcal{T} \setminus \{\alpha_{13}, \alpha_4\}, \mathcal{T} \setminus \{\alpha_1, \alpha_{13}\}, \mathcal{T} \setminus \{\alpha_{12}, \alpha_{13}\}\}$

Tagging Axioms and Concept Pinpointing
We employ the extension of the tableau algorithm which has been developed in [6]. This allows us to capture the parts of axioms responsible for a clash. The concepts or relations in axioms, which are relevant to the clash, are *tagged* and denoted as $tag(C)$. For example, given a set of axioms: $\{A^* \sqsubseteq B^* \sqcap D^* \sqcap E, B^* \sqsubseteq F^* \sqcap G, F^* \sqsubseteq \neg D^*\}$, the concepts with * are relevant to the clash of A. For details of tagging axioms, see [6].

A clash happens when an individual belongs to a concept and to its complement. It is useful to record which concept and its complement is causing the clash. To do this we employ the *concept pinpointing* technique from [13]. This calculates a set of generalised incoherence-preserving terminologies of a TBox $\mathcal{T}$, which is denoted as $git(\mathcal{T})$. Each member of this set is a generalised version of the terminology $\mathcal{T}$, which is generalised as much as possible, while still preserving incoherence. The generalisation works by making all but one axiom trivial (e.g. to make α_5 trivial: UGStudent $\sqsubseteq \top$), and the remaining non-trivial axiom is generalised just to the point where any further generalisation would remove the clash (See [13] for details). In our example, ignoring trivial axioms, $git(\mathcal{T})$ = {{PhDStudent $\sqsubseteq \forall$ take.UGCourse $\sqcap \exists$ take.$\neg$PGCourse}, {MScStudent $\sqsubseteq \forall$ take.UGCourse $\sqcap \exists$take.$\neg$PGCourse }, {MScStudent $\sqsubseteq \forall$take.Exam $\sqcap \exists$take.$\neg$Exam} }. The left hand side of each axiom in a git is a concept name A which is unsatisfiable, and the right hand side gives the concept C and its complement $\neg C$ which cause the clash. From this, we can define $conflict(A)$ as the set of concepts $C, \ldots$ involved in the conflict of some concept A. If $C \in conflict(A)$, this would mean that $A \sqsubseteq C \sqcap \neg C$. Returning to our example again, $conflict$(PhDStudent) is {$\forall$take.PGCourse}, $conflict$(MScStudent) is {$\forall$take.PGCourse, $\forall$take.Exam}.

Maximally Concept-Satisfiable Sub-ontologies (MCSSs)
The algorithm for finding A-MSSs is then used to find MCSSs. A subset $\mathcal{T}'$ of $\mathcal{T}$ is an MCSS of $\mathcal{T}$ iff it is concept-satisfiable, and every $\mathcal{T}''$ such that $\mathcal{T}' \subset \mathcal{T}'' \subseteq \mathcal{T}$ is concept-unsatisfiable. We abbreviate the set of all the MCSS of $\mathcal{T}$ by $MCSS(\mathcal{T})$. In our example, the MCSSs will be the following:

$MCSS(\mathcal{T}) = \{\mathcal{T} \setminus \{\alpha_1, \alpha_{13}\}, \mathcal{T} \setminus \{\alpha_1, \alpha_{10}\}, \mathcal{T} \setminus \{\alpha_1, \alpha_8\}, \cdots\}$

Let $\mathcal{M}$ be a member of $MCSS(\mathcal{T})$. The set of axioms being excluded in $\mathcal{M}$ is denoted as $Ex(\mathcal{M})$. For example $Ex(\mathcal{T} \setminus \{\alpha_1, \alpha_{13}\}) = \{\alpha_1, \alpha_{13}\}$. Note that the size of the exclusion set is the number of axioms being excluded in order to make a consistent sub-ontology. One may prefer to exclude the least possible number of axioms in order to resolve the problem with minimal side effects. In our running example, some sub-ontologies, such as $\mathcal{T} \setminus \{\alpha_1, \alpha_{13}\}$, exclude the minimal number of axioms (two), however, it does not necessarily follow that such sub-ontologies are the best results for users. For example, though excluding α_1 and α_{13} requires a minimal change, a user may strongly believe that the disjointness of UGCourse and PGCourse should be kept (α_1). Besides the size of the exclusion sets of the members of $MCSS(\mathcal{T})$, we also propose knowledge heuristics to analyse the structure and usage of an ontology, and suggest to the user which axioms to select for removal or rewriting.

3 Confidence of Axioms

In an inconsistent/incoherent ontology, some axioms are likely to be correct, while some are likely to be wrong. For example, axioms which are frequently updated are likely to be vulnerable to further modifications, as the users may have been making experimental changes. The *confidence* value indicates how confident we are of the correctness of the axioms in an ontology; we want to exclude the axioms with the least confidence and preserve the ones with the highest confidence. We will describe how to calculate the confidence of individual axioms in detail below, but first we look at how this confidence measure can be used to select an MCSS.

Table 2. Calculating the confidence of a MCSS

1. Given: $\mathcal{M} \in MCSS(\mathcal{T})$,
2. and a set of A_j-MSSs for $j = 1 \ldots n$, where each A_j is an unsatisfiable concept
3. for each $\alpha \in Ex(\mathcal{M})$
4. for each $mss(A_j)$, for $j = 1 \ldots n$, where each A_j is an unsatisfiable concept
5. for each $\mathcal{M}' \in mss(A_j)$
6. if $\alpha \notin \mathcal{M}'$
7. for each $C \in \mathit{conflict}(A_j)$
8. conf += $w_{path} \cdot \mathit{conf}_{path}(\alpha, A_j, C)$;
9. end for
10. break;
11. end if
12. end for
13. end for
14. $\mathit{conf} += w_{sib} \cdot \mathit{conf}_{sib}(\alpha) + w_{disj} \cdot \mathit{conf}_{disj}(\alpha) + w_{usage} \cdot \mathit{conf}_{usage}(\alpha)$;
15. end for
16. conf += $w_{size} \cdot |Ex(\mathcal{M})|$;
17. return conf;

3.1 The Confidence of an MCSS

Recall that $MCSS(\mathcal{T})$ is a set of maximally satisfiable sub-ontologies of the incoherent ontology $\mathcal{T}$. This section describes our approach to selecting the best sub-ontology from $MCSS(\mathcal{T})$ to use. We do this by calculating the confidence of each exclusion set. The algorithm for calculating the confidence of an exclusion set is given in Table 2; lines 3-13 loop through each axiom α in the exclusion set, to find the concepts which clash as a result of α, and to calculate the confidence of the paths on which the clash occurs ($conf_{path}$ is defined in detail in section 3.3). In line 14 the above "confidence of paths" is summed with the confidence value obtained by using three confidence heuristics; these are summed for each axiom in the exclusion set. We will outline these three heuristics for evaluating the confidence of axioms in the next section, and it is feasible that more may be added in the future. From these, a single overall confidence for an axiom may be defined as an aggregation of the heuristics. We use an aggregation function to combine the confidence value of all individual axioms and the size of the set, this aggregation function could be a simple summation, or could assign a weight w_i to each heuristic i, reflecting its importance, where $i \in \{path, sib, disj, usage\}$. In line 16 the size of the exclusion set is also considered because smaller sets are generally preferred, as fewer axioms will need to be excluded. The relative importance of this consideration is captured by w_{size}. Finally the MCSS with the lowest confidence for its exclusion set is recommended to the user.

3.2 Definition of the Confidence of Axioms

We can represent the confidence of an axiom by means of the following function:

Definition 1 *Let α be an axiom in an ontology $\mathcal{O}$, the confidence of α is a function*

$$confidence : \alpha \rightarrow [-1, 1]$$

The confidence function provides a ranking over the axioms, and thus allows us to indicate the correctness of axioms in an orderly way. The confidence of all axioms is presumed to be 0 initially; positive confidence values which are close to 1 indicate axioms which are likely to be "correct", and which should be preserved; negative confidence values which are close to -1 indicate axioms which are likely to be "incorrect", and which should be removed from the ontology. The knowledge heuristics below allow us to evaluate the confidence of axioms through an analysis of (i) the ontology structure and (ii) the patterns of usage in the ontology.

3.3 Knowledge of Ontology Structure

Knowledge of the ontology structure can help us to evaluate the confidence we have in an axiom. We describe here three heuristics which can be used to evaluate the confidence of axioms based on the ontology structure; we call these *structural heuristics*.

(1) Syntactic Relevance Measurement.

Chopra *et al.* [2] propose that the relevance between two formulas in a belief set is dependent on the syntactical structure of the formulas. That means two formulas are relevant to each other if they share an atom. We translate this idea to ontologies by saying that concepts are relevant to each other if they appear on the left hand side and right hand side of the same axiom. Furthermore, we view this relevance as transitive, to a degree, so that two concepts which are connected by a long chain of axioms are less relevant than those connected by a short chain.

Definition 2 *Given two concept names or relation names e_1 and e_2 in $\mathcal{T}$, they are directly relevant, denoted as rel(e_1, e_2), if and only if there is an axiom $\alpha \in \mathcal{T}$ such that e_1 appears on the left hand side of α, e_2 appears on the right hand side.*

Definition 3 *Given two concepts C_1 and C_2 in $\mathcal{T}$, paths(C_1, C_2) is defined as the set of all paths connecting C_1 and C_2. Each path consists of a set of axioms connecting C_1 and C_2, such that rel(C_1, C_3), $\cdots$, rel(C_n, C_2). The strength of paths(C_1, C_2) is defined as*

$$strength(C_1, C_2) = \sum_{path_i \in paths(C_1, C_2)} (1/|path_i|)$$

Where $|path_i|$ is the number of axioms involved in $path_i$.

Definition 4 *Given an unsatisfiable concept A and αin $\mathcal{T}$, and a $C \in conflict(A)$, this means that $A \sqsubseteq C$ and $A \sqsubseteq \neg C$. We compare the relative strength of $paths(A, C)$ with $paths(A, \neg C)$. Path confidence is defined as*

$$conf_{path}(\alpha, A, C) = -strength(A, \neg C)/(strength(A, C) + strength(A, \neg C))$$

Note that in this situation we are dealing with conflicting axioms, this means that at least one will definitely need to be removed. For this reason we are calculating how likely each one is to be removed. If removal is extremely unlikely, then the confidence value should be close to zero, but if removal is likely then the confidence value must be negative. This explains why the result of the above calculation should lie in the range $[-1, 0]$. Roughly speaking, the confidence we have in the subsumption of A by C is given by the confidence of $paths(A, C)$ compared to $paths(A, \neg C)$. For example, the concept PhDStudent is a subconcept of both $\forall$take.PGCourse and its complement.
$paths$(PhDStudent, $\forall$take.PGCourse) =$\{\{\alpha_{11}\}\}$,
$paths$(PhDStudent, $\neg\forall$take.PGCourse) = $\{\{\alpha_4, \alpha_7, \alpha_1, \alpha_{12}\}\}$,
$conf_{path}(\alpha_{11}$,PhDStudent, $\forall$take.PGCourse) = $\frac{-1/4}{1+1/4} = -1/5$;
$conf_{path}(\alpha_4$,PhDStudent, $\neg\forall$take.PGCourse) = $\frac{-1}{1+1/4} = -4/5$.

(2) Comparison between Sibling Concepts.

This heuristic analyses the hierarchical structure of an ontology in two ways: (i) the relationships which concepts participate in. Since all the siblings in the concept hierarchy must be at the same level of generality [11], they usually participate in similar relationships. (ii) disjointness between sibling concepts. All the direct siblings in a well-defined subsumption hierarchy should be disjoint [3]. The confidence of the axioms of a concept can be evaluated by comparing with the concept's siblings. If an axiom α relates a concept C to a relation R, and its siblings are also related to R, then its confidence value will be higher.

Definition 5 *Let $\alpha \in \mathcal{T}$ be an axiom connecting a concept C to a relation R. The siblings of C are denoted by $siblings(C)$; the siblings which are directly relevant to R are denoted by $siblings_R(C)$, i.e. $siblings_R(C) = \{s \in siblings(C) \wedge rel(s, R)\}$. The confidence of α is defined as*

$$conf_{sib}(\alpha) = \frac{|siblings_R(C)|}{|siblings(C)|}$$

The result of the above confidence calculation will be in the range $[0, 1]$; higher values represent axioms that are more likely to be preserved. In our running example, PhDStudent has siblings MScStudent and MPhilStudent which are directly relevant to the relation take, therefore, $conf_{sib}(\alpha_{11}) = 1$.

Furthermore, in a well-modeled ontology the direct siblings, i.e. children of a common parent in the subsumption hierarchy should be disjoint [3].

Definition 6 *Let an axiom α be of form $C_1 \sqsubseteq \neg C_2$ in $\mathcal{T}$, where $C_2 \in siblings(C_1)$; the siblings which are disjoint with C_1 are denoted by $siblings_{disj}(C_1)$. The confidence of α is defined as*

$conf_{disj}(\alpha) = |siblings_{disj}(C_1)|/|siblings(C_1)|$

Therefore, in our running example, the confidence of α_1 is 1, because UGCourse is disjoint with its sibling PGCourse.

3.4 Usage Heuristics

Different ontology engineers may have different views on which coherent sub-ontology should be selected. These views are affected by commonsense knowledge, personal preferences, subjective opinions on the domain, etc. To anticipate the ontology user's perspective on an ontology, we propose a heuristic which can be used to evaluate the confidence of axioms based on knowledge of the history of a user's interaction with the ontology and the reliability of the information source; we call these *usage heuristics*. Firstly, we believe that an axiom's likelihood of being modified in the future depends on the frequency with which it has been modified in the past. We therefore gather the historical knowledge by tracking the users' interactions with the ontology in a log file. All activities the users have performed are recorded in the usage log. Secondly, the effect of the new information is dependent on the reliability of sources, we therefore allow the user to input reliability values, which are -1, 0, or $+1$, on axioms when modifications are made. Here we assume the users rate the reliability of the new information. The "reliable" information has rating $+1$; the "unreliable" information has rating -1; 0 indicates unrated information.

Definition 7 *Let α be an axiom, the reliability rating of the information source of the axiom is defined as*

$$r : \alpha \rightarrow \{-1, 0, +1\}$$

The interplay between time, confidence and reliability provides the following three mutually exclusive cases:

– *Case 1: the information is reliable,* $r = +1$
  In this case it is assumed that newer information (newly added or modified axioms) generally reflects a more accurate view of the domain (The Principle of Primacy of New Information [4]). That means the axioms which are newly added or modified are usually more accurate than those provided earlier and have higher confidence.
– *Case 2: the information is unreliable,* $r = -1$
  In this case the effect of recency is the inverse of the above. The new information can be rejected, because the new information is obtained from unreliable and untrustworthy sources. That means the newly added axioms have lower confidence, and older axioms, having stood the test of time, have higher confidence.
– *Case 3: the information is unknown,* $r = 0$
  If the reliability of the information source is unknown, then the confidence of axioms is dependent on the frequency of modifications, rather than its reliability. Axioms which are frequently updated are likely to be vulnerable to further modifications, as the users may have been making experimental changes. Therefore, the confidence of an axiom is inversely proportional to the frequency of its modifications.

Definition 8 *Let* t *be the length of time since the last modification on an axiom* α*,* r *be the reliability value of the information provided by users, and* $modify(\alpha)$ *be the number of modifications on the axiom* α*,* e^x *be the exponential function. The confidence of the axiom is defined as*

$$conf_{usage}(\alpha) = \begin{cases} \frac{1}{modify(\alpha)\cdot e^t+1} - 1 & \text{if } r = +1, \\ e^{-t} \cdot (\frac{1}{modify(\alpha)+1} - 1) & \text{if } r = -1, \\ \frac{1}{modify(\alpha)+1} - 1 & \text{if } r = 0. \end{cases}$$

The result of the above confidence will be in the range $[-1, 0]$.

3.5 Applications

We now describe some applications of our heuristics. Ontology editors, such as Protégé, KAON etc. log changes and offer users support to annotate these changes, and the changes are logged. Particularly, KAON [5] supports users to rate the importance of ontology elements explicitly. Such information is especially useful in our usage heuristic, as we can make use of the annotations and ratings to select appropriate axiom(s) for modification. In addition, the interplay between time, confidence and reliability are applicable in different scenarios. Firstly, an ontology which describes a certain domain of interest inevitably evolves in the course of its lifetime. Changes in an ontology may be made when the domain of interest changes, the user requirements change, or new information is available for extension. In such cases, the newly added axioms usually have higher reliability, and hence we have higher confidence in them (See case 1 in Definition 8). Secondly, in collaborative ontology building scenarios, it is reasonable to assign higher confidence to the local axioms over axioms from imported ontologies. With the owl:imports construct, we cannot import a certain interesting part of ontology

and leave out the irrelevant part. Therefore, the imported axioms which are contradictory to the existing ones have low reliability and, hence, we have lower confidence in them (See case 2 in Definition 8).

4 Strategies for Rewriting Axioms

After selecting a coherent sub-ontology based on the confidence we have in the axioms, ontology users would like to revise the excluded axioms and put them back into the sub-ontology, rather than to directly remove them from the ontology. To do this, we propose strategies for rewriting axioms in this section, a set of candidate changes is presented to the user, in which the concepts can be either generalised or specialised. We want to achieve the minimal impact of changes; we achieve this by sorting the rewriting suggestions returned to the user, to give preference to the most specific generalisations (or the most general specialisations).

4.1 Generalisation and specialisation

Our strategy for rewriting is to weaken excluded axioms so that the contradiction is removed, however, this can be done in many ways. It is impractical for a system to list all possibilities exhaustively to users. We can weaken an axiom either by making its right hand side more general, or by making its left hand side more specific. We will give some algorithms for *generalising* and *specialising* concepts. In the running example, assume a MCSS, $\mathcal{T} \setminus \{\alpha_{12}, \alpha_{13}\}$, is selected, and we have to rewrite the excluded axioms and put them back into the MCSS. For each axiom in the exclusion set, we have to know which concepts it makes unsatisfiable. In this case we know α_{12} causes PhDStudent to be unsatisfiable, because it is excluded from one of the members of mss(PhDStudent); $conflict$(PhDStudent) is $\{\forall$ take.PGCourse$\}$ (cf. line 3-9 in Table 3). The next step is to generalise and specialise the concepts in α_{12} by the function $weaker$ and $stronger$ respectively. For brevity Table 3 omits some stages: In order to find the *minimal impact* changes, the sets of concepts are sorted w.r.t. subsumption, the most specific one in weaker concepts comes first; the most general one in stronger concepts comes first. Axioms must then be constructed from the sorted concepts, i.e. to put the weakened concepts w_i back into the form $D \sqsubseteq w_i$ and the general ones s_i back into the form $s_i \sqsubseteq E$. Finally a list of sorted candidate changes are returned to the user.

Because of the monotonicity of OWL-DL, weakening axioms in an ontology will not add a new entailment, and so will not cause other concepts in the ontology to be unsatisfiable.

In the following we give an overview of how we generalise and specialise concepts; these methods will then be formalised in our algorithms below.

Generalisation involves replacing the conflicting concept E with a weaker concept expression E' (i.e. less specific $E \sqsubseteq E'$). This can be done by (1) replacing E with its one of its superconcepts, (2) replacing its relation filler with weaker concepts, (3) changing a conjunct to a disjunct, or (4) specialising negated concepts (see Table 4).

Specialisation involves replacing the conflicting concept D with a stronger concept expression D' (i.e. more specific $D' \sqsubseteq D$). This can be done by (1) replacing D with

Table 3. Rewriting an axiom algorithm

1. Given: $\mathcal{M} \in MCSS(\mathcal{T})$,	an $\alpha \in Ex(\mathcal{M})$, where α is of form $D \sqsubseteq E$
2. and a set of A_j-MSSs for $j = 1 \ldots n$,	where each A_j is an unsatisfiable concept
3. $\boldsymbol{C} := \emptyset$	
4. for each $mss(A_j)$, for $j = 1 \ldots n$,	where each A_j is an unsatisfiable concept
5. for each $\mathcal{M}' \in mss(A_j)$	
6. if $\alpha \notin \mathcal{M}'$	
7. $\boldsymbol{C} := \boldsymbol{C} \cup conflict(A_j)$; break;	// A_j has conflict with $\boldsymbol{C}$
8. end if	
9. end for	
10. end for	
11. $w_{set} := weaker(E, \mathcal{T}, A_j, \boldsymbol{C})$;	//a set of generalisation changes
12. $s_{set} := stronger(D, \mathcal{T}, A_j, \boldsymbol{C})$;	//a set of specialisation changes
13. return w_{set}, s_{set};	//return a set of candidate changes

one of its subconcepts, (2) adding a conjunct (e.g. exception of subsumption), (3) replacing its relation filler with stronger concepts, (4) changing a disjunct to a conjunct, or (5) generalising negated concepts (see Table 5).

Table 4. Weaker concepts algorithm

1. Given: A, **C**, and E in $\mathcal{T}$
2.Function: $weaker(E, \mathcal{T}, A$,**C**)
3. **E** := {};
4. if $tag(E)$, then **E** := **E** $\cup superconcepts(E)$;
5. if E is of form $\forall R.F$, and $tag(R)$, then $\forall F' \in weaker(F, \mathcal{T}, A$,**C**), **E** := **E** $\cup\{\forall R.F'\}$;
6. if E is of form $\exists R.F$, and $tag(R)$, then $\forall F' \in weaker(F, \mathcal{T}, A$,**C**), **E** := **E** $\cup\{\exists R.F'\}$;
7. if E is of form $F \sqcap\ G$, and ($tag(F)$ or $tag(G)$), then begin
8. **E** := **E** $\cup\{F \sqcup\ G\}$;
9. if $tag(F)$, then $\forall F' \in weaker(F, \mathcal{T}, A$, **C**), **E** := **E** $\cup\{F' \sqcap\ G\}$;
10. if $tag(G)$, then $\forall G' \in weaker(G, \mathcal{T}, A$, **C**), **E** := **E** $\cup\{G' \sqcap\ F\}$.
11. end if
12. if E is of form $\neg F$ and $tag(F)$, then $\forall F' \in stronger(F, \mathcal{T}, A$, **C**), **E** := **E** $\cup\{F'\}$
13. for each $E_i \in$ **E**
14. for each $C_j \in$ **C**
15. if $E \sqsubseteq \bot$ and $E_i \sqsubseteq \bot$, then **E** := **E** $\setminus\{E_i\}$;
16. else if ($E \sqsubseteq C_j$) and ($E_i \sqsubseteq C_j$), then **E** := **E** $\setminus\{E_i\}$;
17. else if ($E \sqsubseteq \neg C_j$) and ($E_i \sqsubseteq \neg C_j$), then **E** := **E** $\setminus\{E_i\}$;
18. end for
19. end for
20.return **E**;

We now show how Table 4 can be used to generate weaker concepts. The axiom α_{12}: Student $\sqsubseteq \exists$ register.Dept $\sqcap \exists$ take*.UGCourse* is chosen for generalisation. The input concept is of form $F \sqcap\ G$, so we use $line$ 7 in Table 4. Firstly, according to $line$ 8, the intersection restriction can be changed to be union. This gives op_1 below. Next we use $line$ 10, because $\exists$ take*.UGCourse* is tagged. We now have a nested call to $weaker(\exists$ take*.UGCourse$^*, \ldots)$, and in the nesting we firstly use $line$ 4; its weaker concept is $\{\top\}$. This gives op_2. Next we use $line$ 6. As take is tagged, the weaker concepts of its relation filler will be retrieved in a further nested call. In this innermost

Table 5. Stronger concepts algorithm

1. Given: Given: A, **C**, and D in $\mathcal{T}$
2. Function: $stronger(D, \mathcal{T}, A$,**C**)
3. if ($D == A$) return null;
4. **D** := {};
5. if $tag(D)$, then **D** := **D** $\cup subconcepts(D)$;
6. if $tag(D)$ and $A \sqsubseteq D$, then
7. $\forall G \in superconcpet(A) \cup (\{A\} \setminus (superconcept(D) \setminus \{D\}))$, **D** := **D** $\cup\{D \sqcap \neg G\}$;
8. if D is of form $\forall R.F$, and $tag(R)$, then $\forall F' \in stronger(F, \mathcal{T}, A$,**C**), **D** := **D** $\cup\{\forall R.F'\}$;
9. if D is of form $\exists R.F$, and $tag(R)$, then $\forall F' \in stronger(F, \mathcal{T}, A$,**C**), **D** := **D** $\cup\{\exists R.F'\}$;
10. if D is of form $F \sqcup G$, and ($tag(F)$ or $tag(G)$), then begin
11. **D** := **D** $\cup\{F \sqcap G\}$;
12. if $tag(F)$, then $\forall F' \in stronger(F, \mathcal{T}, A$, **C**), **D** := **D** $\cup\{F' \sqcup G\}$;
13. if $tag(G)$, then $\forall G' \in stronger(G, \mathcal{T}, A$, **C**), **D** := **D** $\cup\{G' \sqcup F\}$.
14. end if
15. if D is of form $\neg F$ and $tag(F)$, then $\forall F' \in weaker(F, \mathcal{T}, A$, **C**), **D** := **D** $\cup\{F'\}$
16. for each $D_i \in$ **D**
17. if $A \sqsubseteq D_i$, then **D** := **D** $\setminus\{D_i\}$;
18. end for
19. return **D**;

nesting we note that UGCourse is tagged and so we use $line$ 4; its weaker concepts are $\{\top$, Course$\}$. This gives op_3 and op_4. Finally, there are four weaker concepts: op_1: $\exists$ register.Dept $\sqcup \exists$ take.UGCourse; op_2: $\exists$ register.Dept; op_3: $\exists$ register.Dept $\sqcap \exists$ take.$\top$; and op_4: $\exists$ register.Dept $\sqcap \exists$ take.Course. The four candidate changes provided for the axiom α_{12} are shown in Figure 1C.

4.2 Application to examples

Addressing the two typical scenarios in Section 1, we are now able to provide support for rewriting the axioms. In the first case, assume Lion $\sqsubseteq \exists$ eats (Sheep $\sqcap$ Cow) is chosen to be rewritten, the relation filler can be generalised to be Sheep $\sqcup$ Cow (see Table 4). Therefore, our algorithms can guide the users to pick appropriate operations and minimise the impact of changes. In the second case, assume that Bird $\sqsubseteq$ CanFly is chosen for modification. We can specialise the concept Bird such that all birds but penguin can fly, i.e. Bird $\sqcap \neg$ Penguin $\sqsubseteq$ CanFly (see Table 5).

5 Related Work

To our knowledge, the most relevant work is SWOOP [7], where the authors propose strategies used to *rank* erroneous axioms in order of importance and rewrite them. They propose various strategies to rank the erroneous axiom(s) to be removed from the MUPS in order to fix the unsatisfiable concepts. We now compare their strategies with our approach:

1. Provenance information (where precedence means the status or authority of the user) –The authors of SWOOP mention that the change-log of ontologies can be

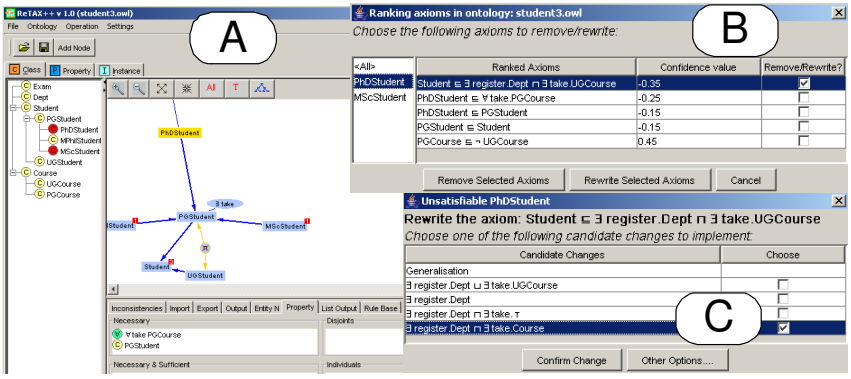

Fig. 1. (A) The screenshot of the implemented tool, in which an ontology is detected with unsatisfiable concepts. (B) The set of excluded axioms are ranked based on the confidence value. (C) A list of sorted candidate changes are provided.

shared, and that axioms added by a user with a high precedence level will be given high importance. However, the details are sketchy in their paper. We propose to utilise the usage-log of an ontology to analyse the frequency of modifications with respect to the time-stamp and the reliability of the information source.

2. Relevance to the ontology in terms of its usage – in SWOOP the relevance of an element to the ontology depends on the usage of the element in its ontology. That means, if a concept is highly referenced by other elements in an ontology, then it is more relevant to the ontology, and hence more important. This strategy is different to our structural heuristic which places more emphasis on ranking axioms according to their likelihood to be correct, rather than the ramifications of removing them.
3. Impact on ontology when an axiom is removed or modified – SWOOP ranks axioms based on the number entailments which would be broken if the axioms were removed. This implies that an axiom which breaks more relationships is more important in the ontology. However, we consider this point not at the ranking phase, but in our strategies for rewriting axioms. Our algorithms generate a set of candidate changes and give preference to the changes which cause the minimal impact of changes.

Few approaches have been proposed concerning the strategies for resolving inconsistent ontologies. In [12], the authors propose a set of rules to resolve the detected inconsistency. They weaken restrictions either by removing an axiom, replacing it with its superconcepts, or changing its cardinality restriction values. In contrast, our approach allows users to select from a range of different weakening strategies. On the other hand, the authors of SWOOP propose another approach to resolving inconsistency. They suggest rewriting the axiom to preserve as much information as possible while eliminating unsatisfiability. A library of commonly occurring error patterns is maintained; If any erroneous axiom has a pattern corresponding to one of the common error patterns, then they suggest the intended axiom to the user as a replacement. However, the common error patterns may only apply for those ontologies built by non-expert users, it is in-

sufficient to cover other applications, such as ontology integration. Moreover, by correcting ontologies based on the common error patterns of SWOOP, one may arrive at an inconsistent ontology. In comparison, our approach only weakens axioms, so further inconsistencies will not be introduced by the candidate changes. SWOOP additionally uses the heuristic of the concept's name to infer the relationships. However, it is too simple to infer more complex relationships.

6 Conclusion and Future Work

In this paper we identify two typical scenarios of inconsistent/incoherent ontologies and extend existing approaches by proposing some methods for ranking and rewriting axioms in ontologies. Structural and usage heuristics are formalised to evaluate the confidence of axioms, so that we are able to select a maximal coherent sub-ontology (MCSS) which is most likely to be correct. Furthermore, we propose strategies to rewrite the excluded axioms and put them back into the MCSS, rather than remove them directly. With our approach, users are provided with guidelines on how to achieve a consistent/coherent ontology. We aim to enable even users who are not experts in ontology engineering to resolve inconsistencies. The approach has yet to be evaluated in the context of various ontologies. For future work we plan to implement our prototype as a plug-in in existing tools, such as Protégé. We are currently extending our methods to deal with more expressive description logics as well.

References

1. Tim Berners-lee. Semantic Web Road Map. W3C Design Issues. URL http://www.w3.org/DesignIssues/Semantic.html, Oct. 1998.
2. S. Chopra, R. Parikh, and R. Wassermann. Approximate belief revision. *Logic Journal of the Interest Group in Pure and Applied Logics (IGPL)*, 9(6):755–768, 2001.
3. R. Cornet and A. Abu-Hanna. Evaluation of a frame-based ontology. a formalization-oriented approach. In *MIE2002, Budapest, Studies in Health Technology and Information*, volume 90. G. Surjn , R. Engelbrecht and P. McNair, eds, 2002.
4. M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Seventh National Conference on Artificial Intelligence*, volume 2, pages 475–479, California, 1988. AAAI Press.
5. P. Haase. Incremental ontology evolution: Evaluation. SEKT Formal Deliverable D3.1.2, University of Karlsruhe, December 2005.
6. A. Kalyanpur, B. Parsia, B. C. Grau, and E. Sirin. Tableaux tracing in SHOIN. Technical Report UMIACS-TR 2005-66, University of Maryland, November 2005.
7. A. Kalyanpur, B. Parsia, E. Sirin, and B. Cuenca-Grau. Repairing Unsatisfiable Concepts in OWL Ontologies. In *Proc. of the Third European Semantic Web Conference*, June 2006.
8. H. Katsuno and A. O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1992.
9. SC J. Lam, D. Sleeman, and W. Vasconcelos. Graph-based ontology checking. In *the workshop Ontology Management: Searching, Selection, Ranking, and Segmentation in K-CAP 05*, October 2005.

10. T. Meyer, K. Lee, R. Booth, and J. Z. Pan. Finding maximally satisfiable terminologies for the description logic ALC. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, July 2006.
11. N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory, KSL-01-05, 2001.
12. P. Plessers and O. De Troyer. Resolving inconsistencies in evolving ontologies. In *Proceedings of the Third European Semantic Web Conference (ESWC-2006)*, June 2006.
13. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *8th International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.
14. M. Schmidt-Schauss and G. Smolka. Attribute concept descriptions with complements. *Artificial Intelligence*, 48(0):1–26, 1991.
15. M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.