Coetzer S, Britz K. To appear in: Neuhaus F, Brodaric B, eds. Proceedings of the 12th Int'l Conf. (FOIS 2021), Frontiers in Artificial Intelligence and Applications by IOS Press.

Debugging Classical Ontologies Using Defeasible Reasoning Tools

Simone COETZER and Katarina BRITZ
CAIR, Information Science Dept, Stellenbosch University

Abstract
A successful application of ontologies relies on representing as much accurate and relevant domain knowledge as possible, while maintaining logical consistency. As the successful implementation of a real-world ontology is likely to contain many concepts and intricate relationships between the concepts, it is necessary to follow a methodology for debugging and refining the ontology. Many ontology debugging approaches have been developed to help the knowledge engineer pinpoint the cause of logical inconsistencies and rectify them in a strategic way. We show that existing debugging approaches can lead to unintuitive results, which may lead the knowledge engineer to opt for deleting potentially crucial and nuanced knowledge. We provide a methodological and design foundation for weakening faulty axioms in a strategic way using defeasible reasoning tools. Our methodology draws from Rodler's interactive ontology debugging approach and extends this approach by creating a methodology to systematically find conflict resolution recommendations. Importantly, our goal is not to convert a classical ontology to a defeasible ontology. Rather, we use the definition of exceptionality of a concept, which is central to the semantics of defeasible description logics, and the associated algorithm to determine the extent of a concept's exceptionality (their ranking); then, starting with the statements containing the most general concepts (the least exceptional concepts) weakened versions of the original statements are constructed; this is done until all inconsistencies have been resolved.

Keywords. knowledge representation and reasoning, formal ontology, ontology debugging tools, defeasible description logic

1. Introduction

With ontologies becoming increasingly important across many industries (see for instance [1,2,3]), the need for a strategic debugging methodology is clear. Furthermore, ontologies are being applied to more abstract and complex domains such as business and law. Indeed, in the FOIS 2020 selected papers, we saw the prevalence of topics relating to these realms (see for instance [4,5,6]). Often in these domains we find that knowledge is described by referring to typical properties of a specific concept, but for specific cases the general theory can be overwritten. Arguably, humans naturally employ a method of non-monotonic reasoning when building up knowledge: facts are assumed to be correct until an exception to the facts is encountered – the Quinean web of knowledge is then slightly adjusted [7]. This highlights that not only is a *strategic* debugging methodology necessary, but that the debugging methodology should be able to deal with *logical nuances*.

Multiple debugging tools have been developed precisely so that a more strategic methodology for pinpointing the root cause faulty axioms is established (see for instance [8,9,10]). Notably, recently Schekotikin, et al. [11] created an interactive ontology debugging tool. This methodology has been instantiated in a Protégé tool, *OntoDebug*, in which the queries are methodically and iteratively posed to the user until a single diagnosis is identified, at which point the user can then make a repair.

This approach is a step in the right direction to guiding the user in the debugging process. However, in the case where what we call 'multi-level exceptions' occur this approach can sometimes lead to unintuitive results. Consider for instance the following *incoherent* ontology (i.e., some concepts are unsatisfiable, and cannot be instantiated in any model of the ontology):

$$\mathscr{O} = \left\{ \begin{array}{l} \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo}.\mathsf{ConfidentialInfo} \\ \mathsf{User} \sqsubseteq \exists \mathsf{accessTo}.\mathsf{PublicInfo} \\ \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo}.\mathsf{ConfidentialInfo} \\ \mathsf{BlackListedStaff} \sqsubseteq \mathsf{Staff} \\ \mathsf{BlackListedStaff} \sqsubseteq \neg\exists \mathsf{accessTo}.\mathsf{ConfidentialInfo} \end{array} \right\}$$

When the OntoDebug tool is run, the suggestion is to remove the axiom stating that Staff have access to some ConfidentialInfo. Indeed, from a classical ontology perspective, the suggestion of this repair makes sense – we have asserted that a User (the concept subsuming Staff) does not have access to some ConfidentialInfo, and we have also asserted that BlacklistedStaff (the concept subsumed by Staff) does not have access to some ConfidentialInfo.

Yet, intuitively we know that there are exceptions to the rule, and that Staff is an exception to the more general concept of User, and that BlacklistedStaff is the exception to the concept of Staff. In this case, the intuition that we would like to maintain the User-Staff-BlackListedStaff hierarchy stems from the fact that User has access to PublicInfo, a property which we would like Staff and BlackListedStaff to inherit. We know, therefore, that a more accurate repair would be to change the first and fourth axioms to read semantically as follows: User *usually* does not have access to some ConfidentialInfo; Staff *usually* do have access to some ConfidentialInfo.

We therefore propose an extension to the interactive ontology debugging methodology, which allows for recommendations to be made on how axioms can be weakened rather than deleted to further the goal of knowledge retention. Importantly, our goal is not to convert a classical ontology to a defeasible ontology – therefore we do not use defeasible reasoning support through, for example, the computation of rational closure. Rather, we use the definition of exceptionality of a concept, which is central to the semantics of defeasible description logics, and the associated algorithm to determine the extent of a concept's exceptionality (their ranking); then, starting with the statements containing the most general concepts (the least exceptional concepts) weakened versions of the original statements are constructed; this is repeated until all inconsistencies have been resolved.

Our approach differs from repair strategies that remove (parts of) axioms, possibly after computing smaller laconic or precise justifications [12]. Instead, our methodology aims to identify missing parts of axioms and add them.

The remainder of the paper is structured as follows: In Section 2 we present the necessary background to non-monotonic reasoning required for the development of our debugging strategy. Section 3 provides the necessary background on the interactive ontology debugging tool, OntoDebug, that our solution uses as a basis. Then in Section 4, we explain our extension to the interactive ontology debugging tool – this extension allows the user to view recommendations on how faulty axioms can be *weakened* rather than *deleted* through the use of defeasible DL tools. In the final section, we conclude on the contributions of our work, and what the focus of future work could be.

2. Defeasible Description Logics

The notion of defeasibility originates from non-monotonic logics. First described by McDermott and Doyle [13], the notion of non-monotonic logics were formed in contrast to monotonic or classical logics. McDermott and Doyle [13] argue that classical monotonic logics do not take into account that our human knowledge is incomplete and thus, with the addition of new facts, old facts may become invalidated or weakened.

Several non-monotonic extensions of DLs exist [14,15,16]. Britz et al. [17,18] extended the work of Kraus, Lehmann and Magidor (KLM) [19] beyond propositional logics to DLs, and their extension includes an implementation. They provide a semantic account of both preferential and rational subsumption relations based on the standard semantics of description logics. The same benefits that are obtained by using the KLM approach on propositional logic are realised when extending this approach to DLs. The KLM approach provides natural and intuitive semantics for defeasible subsumption. In the context of ontology debugging, the main benefit of using the KLM approach lies in the fact that it allows for defeasible subsumption problems to be reduced to classical entailment checking – this also has the effect that defeasibility can be introduced without increasing the computational complexity associated with classical DL reasoning tasks.

The concept language $\mathscr{L}$ of the description logic $\mathscr{ALC}$ is built according to the following rule:

$$C := \top \mid \bot \mid A \mid \neg C \mid C \sqcup C \mid C \sqcap C \mid \forall r.C \mid \exists r.C$$

Given $C, D \in \mathscr{L}$, $C \sqsubseteq D$ is called a subsumption statement, or general concept inclusion (GCI), read 'C is subsumed by D'. Defeasible subsumption, also referred to as defeasible concept inclusion, is intended as defeasible counterpart of classical subsumption. Given concepts C and D from $\mathscr{L}$ a *defeasible concept inclusion axiom* (DCI, for short) is a statement in the form $C \sqsubset\!\!\!\sim D$.

Statements that are written in the form $C \sqsubset\!\!\!\sim D$ should be read as 'C is *usually* subsumed by D' or 'individuals that are typical C's are also elements of D'. The symbol $\sqsubset\!\!\!\sim$ denoting defeasible subsumption can thus be used in the same way as classical subsumption $\sqsubseteq$, the difference being that it refers to defeasible concept inclusion, and that the inclusion may be violated for exceptional individuals.

As is the case with classical subsumption $\sqsubseteq$, its defeasible counterpart $\underset{\sim}{\sqsubset}$ also acts as a connective positioned between the concept language at the object level, and the meta-language (the level of entailment). The semantics of $\underset{\sim}{\sqsubset}$ is defined formally w.r.t. preferential interpretations. Reasoning tasks can however be reduced to classical reasoning without affecting complexity, and has been implemented as a plugin on the Protégé platform.

The definitions below, and the algorithms to implement basic defeasible reasoning tasks, are discussed and motivated in detail in [18]. Readers who are not familiar with the theoretical foundations of the KLM approach to defeasible reasoning can focus on the intuitive explanations we provide, without engaging with the deeper mathematical foundations. We also assume familiarity with the semantics of standard description logic such as $\mathscr{ALC}$, and build on its syntax and semantics:

Definition 1. *A **preferential interpretation** is a structure $\mathscr{P} := \langle \Delta^{\mathscr{P}}, \cdot^{\mathscr{P}}, \prec^{\mathscr{P}} \rangle$ where $\langle \Delta^{\mathscr{P}}, \cdot^{\mathscr{P}} \rangle$ is a DL interpretation (which we denote by $\mathscr{I}_{\mathscr{P}}$ and refer to as the classical interpretation associated with $\mathscr{P}$), and $\prec^{\mathscr{P}}$ is a strict partial order on $\Delta^{\mathscr{P}}$ (i.e., $\prec^{\mathscr{P}}$ is irreflexive, transitive and asymmetric) satisfying the smoothness condition (for every $C \in \mathscr{L}$, if $C^{\mathscr{P}} \neq \emptyset$, then $min_{\prec^{\mathscr{P}}} (C^{\mathscr{P}}) \neq \emptyset$).*

Definition 2. *A defeasible subsumption relation $\underset{\sim}{\sqsubset}$ is a **preferential subsumption relation** if it satisfies the following set of properties, called the preferential KLM properties for DLs:*

$$(Ref)\ C \underset{\sim}{\sqsubset} C \qquad (LLE)\ \frac{C \equiv D,\ C \underset{\sim}{\sqsubset} E}{D \underset{\sim}{\sqsubset} E} \qquad (And)\ \frac{C \underset{\sim}{\sqsubset} D,\ C \underset{\sim}{\sqsubset} E}{C \underset{\sim}{\sqsubset} D \sqcap E}$$

$$(Or)\ \frac{C \underset{\sim}{\sqsubset} E,\ D \underset{\sim}{\sqsubset} E}{C \sqcup D \underset{\sim}{\sqsubset} E} \qquad (RW)\ \frac{C \underset{\sim}{\sqsubset} D,\ D \sqsubseteq E}{C \underset{\sim}{\sqsubset} E} \qquad (CM)\ \frac{C \underset{\sim}{\sqsubset} D,\ C \underset{\sim}{\sqsubset} E}{C \sqcap E \underset{\sim}{\sqsubset} D}$$

Along with the above properties, if the relation $\underset{\sim}{\sqsubset}$ also satisfies rational monotonicity (RM), then it is a *rational* subsumption relation:

$$(\mathrm{RM})\ \frac{C \underset{\sim}{\sqsubset} D,\ C \not\underset{\sim}{\sqsubset} \neg E}{C \sqcap E \underset{\sim}{\sqsubset} D}$$

Definition 3. *Given $C, D \in \mathscr{L}$, a statement of the form $C \underset{\sim}{\sqsubset} D$ is a defeasible subsumption statement. A preferential interpretation $\mathscr{P} = \langle \Delta^{\mathscr{P}}, \bullet^{\mathscr{P}}, \prec \mathscr{P} \rangle$ **satisfies** a defeasible subsumption statement $C \underset{\sim}{\sqsubset} D$, if $min_{\prec^{\mathscr{P}}} (C^{\mathscr{P}}) \subseteq D^{\mathscr{P}}$.*

It is desirable for the defeasible entailments to adhere to rational monotonicity as it is a prerequisite for the *presumption of typicality* to hold. The presumption of typicality states that all individuals are considered to be most normal unless they are proven to be exceptional. This is central to the notion of a rational preferential ordering.

Preference orders allow individuals or objects (and, by extension, also concepts and statements) to be ordered or ranked based on their level of exceptionality relative to other individuals, concepts or statements in an ontology. In a propositional setting, this takes the form of an ordering on worlds. An object's normality or typicality is determined not by some intrinsic characteristic that the object possesses, but rather in relation to the other objects in the domain. The assumption of rationality (RM) imposes a further restriction on preference orders, namely that they are *modular*. This partitions the domain into layers that are linearly ordered.

Definition 4. *Given a set X, $\prec \subseteq X \times X$ is a* ***modular order*** *if it is a strict partial order, and its associated incomparability relation $\sim$, defined by $x \sim y$ if neither $x \prec y$ nor $y \prec x$, is transitive.*

Definition 5. *A* ***modular interpretation*** *is a preferential interpretation $\mathscr{R} = \left\langle \Delta^{\mathscr{R}}, \bullet^{\mathscr{R}}, \prec \mathscr{R} \right\rangle$ such that $\prec^{\mathscr{R}}$ is modular.*

Definition 6. *A statement α is* ***modularly entailed*** *by a defeasible knowledge base $\mathscr{O}$, written $\mathscr{O} \models_{mod} \alpha$, if every modular model of $\mathscr{O}$ satisfies α.*

However, it turns out that modular entailment represents a monotonic entailment relation, which thus reduces entailment from a knowledge base to classical reasoning. Furthermore, modular entailment is not necessarily rational. In order to obtain a non-monotonic entailment relation that is also rational, we need to look beyond a Tarskian-style consequence relation.

Our focus in this paper is not on rational entailment, but rather on the notion of *exceptionality*, a central building block in the computation of rational closure, a form of rational entailment.

Definition 7. *Let $\mathscr{O}$ be a defeasible knowledge base and $C \in L$. We say C is* ***exceptional*** *in $\mathscr{O}$ if $\mathscr{O} \models_{mod} \top \mathrel{\underset{\sim}{\sqsubset}} \neg C$. A DCI $C \mathrel{\underset{\sim}{\sqsubset}} D$ is exceptional in $\mathscr{O}$ if C is exceptional in $\mathscr{O}$.*

Intuitively, an exceptional concept C w.r.t. a knowledge base $\mathscr{O}$ is one to which no normal individual in the domain of $\mathscr{O}$ can belong. This definition of an exceptional concept is used in [18] to compute the *rank* of a concept. Briefly, the more exceptional a concept is, the higher is its rank. Using the defeasible counterpart of the User-Staff-BlacklistedStaff example from Section 1, the rankings would be computed as follows:

1. First, the left-hand-side concept of all defeasible statements that are non-exceptional (according to Definition 7) are given a ranking of 0. The DCIs with non-exceptional left-hand side concepts are also given a rank of 0. In this case, the concept User is assigned a rank of 0.
2. Then, a new knowledge base is created containing only the remaining exceptional statements along with the classical General Concept Inclusions (GCIs) in the knowledge base. For the left-hand side concepts of defeasible statements that are now deemed to be non-exceptional, a ranking of 1 is given to left hand side concept contained in the axiom. The DCIs with a non-exceptional left-hand side concept are also given a rank of 1. In this case, the concept Staff is assigned a rank of 1.
3. The above procedure from step 2 is repeated and with each iteration, the ranking of the left hand side concept is increased by 1. In this case, the concept BlacklistedStaff is assigned a rank of 2.
4. Once all the DCIs have been ranked, or there are no new non-exceptional concepts in the last step, if there are any concepts that remain they are given a rank of ∞.This means that the concept is, even when preferential ordering has been applied, unsatisfiable. In our example ontology, there are no further statements to assess, and so no concepts are assigned a rank of ∞.

These rankings can then be used to determine what is rationally entailed by a knowledge base:

Definition 8. *$C \mathrel{\underset{\sim}{\sqsubset}} D$ is in the rational closure of a knowledge base $\mathscr{O}$ if*

$$rank(C \sqcap D) < rank(C \sqcap \neg D) \text{ or } rank(C) = \infty.$$

Intuitively, this definition states that no normal object can belong to C but not to D. Such objects must be the exception in any preferential model.

3. Interactive Ontology Debugging

Basic ontology debugging focuses on finding justifications for inconsistencies in a faulty ontology. Although the basic concepts assist with fault *identification* in ontologies, an exponential number of minimal conflict sets may exist for the exceptions in an ontology. Thus, there is a need for fault *localisation* – i.e. not returning all axioms from all conflict sets, but presenting the user with only the axiom(s) that represent the root cause of the problem. In the ontology debugging community, then, it has been suggested that background knowledge, along with positive and negative test cases, should be explicitly provided as input by the user so that the test cases along with the background knowledge eliminate some of the axioms that are returned in the minimal conflict set [11].

Definition 9. *Let $\mathscr{O}$ be an ontology, and let $\mathscr{B} \subseteq \mathscr{O}$ be the* background knowledge *to $\mathscr{O}$. Then all axioms in $\mathscr{B}$ are assumed to be correct. In the context of ontology debugging, the remainder of axioms in $\mathscr{O}$ are considered potentially faulty [20].*

Background knowledge constitutes axioms that the oracle or knowledge engineer knows to be true before starting with testing. In the OntoDebug tool, the dialogue on background knowledge gets populated by the Abox statements. In the absence of Abox statements, Abox statements are auto-generated for each concept.

Positive and negative test cases are usually formulated once the knowledge engineer or oracle starts with their testing, and through the testing they uncover:

- axioms that they do not want to exist in future (negative test cases), or
- axioms that they do want to exist in future, but which were at a stage in testing not present (positive test cases).

Definition 10. ***Positive test cases*** *(aggregated in the set P) correspond to desired entailments of the correct (repaired) ontology, $\mathscr{O}$ along with the background knowledge $\mathscr{B}$. Each test case $p \in P$ is a set of axioms over language $\mathscr{L}$. The meaning of a positive test case $p \in P$ is that some axiom p (or the conjunction of axioms P in the case of a set of p) must be entailed by the correct $\mathscr{O}$ integrated with $\mathscr{B}$ [20].*

Definition 11. ***Negative test cases*** *(aggregated in the set N) represent undesired entailments of the correct (repaired) ontology $\mathscr{O}$, along with the background knowledge $\mathscr{B}$. Each test case $n \in N$ is a set of axioms over language $\mathscr{L}$. The meaning of a negative* *test case $n \in N$ is that some axiom n (or the conjunction of axioms N in the case of a set of N) must not be entailed by the correct $\mathcal{O}$ integrated with $\mathcal{B}$ [20].*

Once background knowledge, and positive and negative test cases are provided for the ontology, this is put together in a diagnosis problem instance (DPI) which gives the parameters in which the diagnosis should be calculated.

Definition 12. *Let $\mathcal{O}$ be an ontology (including possibly faulty axioms) and $\mathcal{B}$ be background knowledge (including correct axioms) where $\mathcal{O} \cap \mathcal{B} = \emptyset$, and let $\mathcal{O}^*$ denote the (unknown) intended ontology. Moreover, let P and N be sets of axioms where each $p \in P$ must and each $n \in N$ must not be entailed by $\mathcal{O}^* \cup \mathcal{B}$, respectively. Then, the tuple $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ is called a* ***diagnosis problem instance (DPI)*** *[11].*

Definition 13. *Let $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ be a DPI. Then, a set of axioms $\mathcal{D} \subseteq \mathcal{O}$ is a* ***diagnosis*** *if and only if both of the following conditions hold:*

1. *$(\mathcal{O} \setminus \mathcal{D}) \cup P \cup \mathcal{B}$ is consistent (coherent if required)*
2. *$(\mathcal{O} \setminus \mathcal{D}) \cup P \cup \mathcal{B} \not\models n$ for all $n \in N$*

A diagnosis $\mathcal{D}$ is minimal iff there is no $\mathcal{D}' \subset \mathcal{D}$ such that $\mathcal{D}'$ is a diagnosis [11].

If background knowledge, positive and negative test cases are incorporated when diagnoses are determined, this will limit the number of potentially faulty axioms that are output as explicit instructions are given as to which entailments and axioms can be deemed correct or incorrect [20]. Rodler's suggestion is to automate the process of finding test cases by developing an algorithm which, targeting the most likely diagnoses first, iteratively asks the knowledge engineer (in this case, someone who is referred to as the 'oracle' – someone who has full knowledge of a given domain) whether certain axioms should or should not be entailed.

Definition 14. *Let* $\boldsymbol{Ax}$ *be a set of axioms and* $ans : \boldsymbol{Ax} \rightarrow P \cup N$ *a function which assigns axioms in* $\boldsymbol{Ax}$ *to either the positive or negative test cases. Then, we call ans an* ***oracle*** *w.r.t. the intended ontology $\mathcal{O}^*$, iff for each $ax \in \boldsymbol{Ax}$ both the following conditions hold:*

1. *$ans(ax) = P \rightarrow \mathcal{O}^* \cup \mathcal{B} \models ax$*
2. *$ans(ax) = N \rightarrow \mathcal{O}^* \cup \mathcal{B} \not\models ax$*

[11].

A query is a set of axioms which, once the knowledge engineer/ oracle provides an answer as to whether the entailments should hold or not, sufficient information is obtained such that at least one diagnosis can be eliminated.

Definition 15. *Let $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ be a DPI, $\mathcal{D}$ be a set of diagnoses for this DPI, and Q be a set of axioms. Then we call Q a* ***query*** *for $\mathcal{D}$ iff, for any classification Q^P_{ans}, Q^N_{ans} of the axioms in Q of a domain expert oracle ans, at least one diagnosis in $\mathcal{D}$ is no longer a diagnosis for the new DPI $\langle \mathcal{O}, \mathcal{B}, P \cup Q^P_{ans}, N \cup Q^N_{ans} \rangle$ [11].*

The knowledge engineer's answers to these queries are added to the list of test cases. The process of posing queries to the knowledge engineer, and feeding through the knowledge engineer's answer, and recomputing the new diagnoses is performed until only minimal number faulty axioms remain for each DPI.

4. Defeasible Reasoning Support for Interactive Ontology Debugging

As illustrated in the Introduction, multi-level exceptions can lead to unintuitive results and loss of information when axioms are removed while following Rodler's interactive debugging methodology.

We propose that Rodler's [20] original interactive ontology debugging methodology be followed until an unintuitive result is obtained. If the interactive ontology debugging methodology is followed, and we get to an unintuitive suggestion for an axiom to repair, the following methodology is followed:

1. **Isolate the issue:** Create a separate sub-ontology, $\mathcal{O}^{'}$ containing the axiom listed for repair, along with axioms that, from the minimal conflict sets, lead to this axiom being identified as a potentially faulty axiom.
2. **Determine a candidate axiom to weaken, and a candidate weakening concept with which to weaken the candidate axiom:** To determine this, the ranking algorithm is used on the above ontology $\mathcal{O}^{'}$: central to the ranking formula is the notion of exceptionality.
   (a) Ranking of 0 – least exceptional: First we identify the concepts with a rank of 0. In this case this would be User. Then, the statements with a rank of 0 are identified.
   (b) Ranking of 1 – concepts that are exceptional w.r.t. level 0 statements: $\mathcal{O}^{''}$ now contains only the remaining exceptional statements after the axioms that now have an associated ranking have been removed.

   Per the ranking algorithm, concepts now have the following ranking:

World order/ rank	Concept
0	User
1	Staff

Table 1. First iteration concept ranking output.

The axioms are then ranked to correspond to the ranking of the respective LHS concept. Therefore, it follows that the axioms have the following ranking:

World order/ rank	Axiom
0	$\mathsf{User} \sqsubsetsim \neg\exists \mathsf{accessTo.ConfidentialInfo}$
1	$\mathsf{Staff} \sqsubsetsim \mathsf{User}$
1	$\mathsf{Staff} \sqsubsetsim \exists \mathsf{accessTo.ConfidentialInfo}$

Table 2. First iteration axiom ranking output.

It should be noted that even though in a minimal conflict set there may be concepts that are ranked at a level higher than 1, only concepts (and axioms) at levels 0 and 1 will be used in the next step. Furthermore, it is only ever necessary to work on these two levels to systematically resolve multi-level exceptions as the conflicts preceding the next level would have been solved already.

3. **Weaken the relevant axiom:** Next, the postulate of Cautious Monotonicity is applied to weaken the axiom at level 0. As referenced in Definition 2:

$$\text{(CM)}\ \frac{C \sqsubsetsim D,\ C \sqsubsetsim E}{C \sqcap E \sqsubsetsim D}$$

The weakened result we would like to get to has a form similar to that of the axiom below the line: $C \sqcap E \sqsubsetsim D$. In our case, the weakened result would be $\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubsetsim \neg\exists\mathsf{accessTo.ConfidentialInfo}$. Thus we find that in the postulate of Cautious Monotonicity, C can represent User, D can represent $\neg\exists\mathsf{accessTo.ConfidentialInfo}$ and E can represent $\neg\mathsf{Staff}$:

$$\text{(CM)}\ \frac{\mathsf{User} \sqsubsetsim \neg\exists\mathsf{accessTo.ConfidentialInfo},\ \mathsf{User} \sqsubsetsim \neg\mathsf{Staff}}{\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubsetsim \neg\exists\mathsf{accessTo.ConfidentialInfo}}$$

The rule that is extrapolated here is thus that when using Cautious Monotonicity to apply weakening to an axiom at level 0, use the axiom as is for the first premise (top left axiom) in the postulate; for the second premise (top right axiom), use the subsumed (left hand) concept at level 0 subsumed by the negation of a concept at level 1; the resultant conclusion (bottom axiom) is then the axiom showing the weakened result. This step is mandated by Lemma 1 below.

4. **Choose to accept or reject solution:** The classical counterpart of the defeasible axiom obtained by applying Cautious Monotonicity is what is then displayed to the knowledge engineer as a repair recommendation, and the can choose to accept or reject.

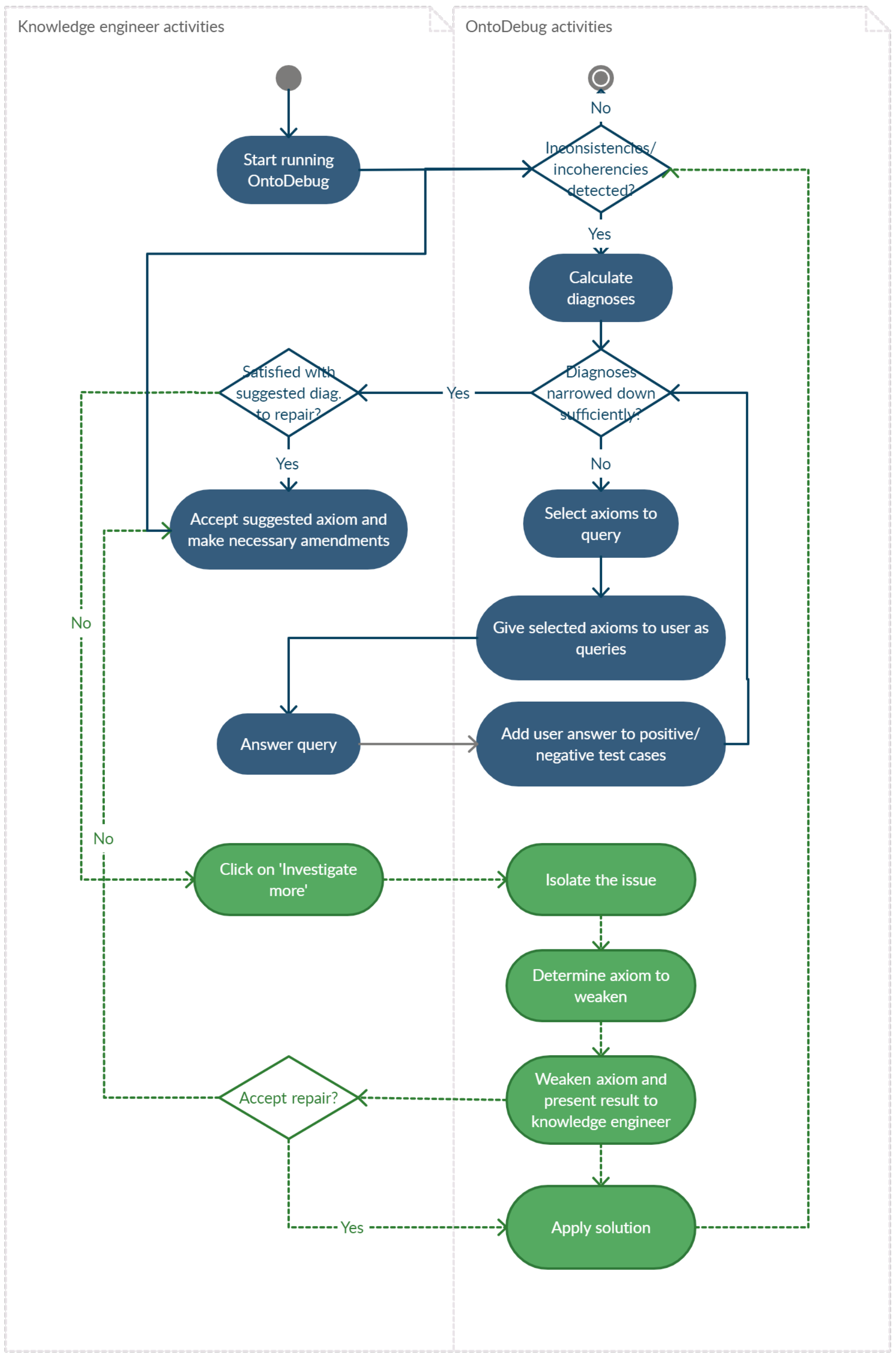
5. **Repeat until done:** This process is repeated until all inconsistencies have been resolved.

Lemma 1. *Let $\mathcal{O}$ be a defeasible knowledge base, and let C and E be concepts with $rank(C) = 0$ and $rank(E) = 1$. It then follows that $C \sqsubsetsim \neg E$ is in the rational closure of $\mathcal{O}$.*

Proof. Since $rank(C) = 0$, it follows that either $rank(C \sqcap E) = 0$ or $rank(C \sqcap \neg E) = 0$. But since $rank(E) = 1$, $rank(C \sqcap E) \geq 1$. Therefore, $rank(C \sqcap \neg E) = 0$, and hence $rank(C \sqcap \neg E) < rank(C \sqcap E)$. It follows from Definition 8 that $C \sqcap \neg E$ is in the rational closure of $\mathcal{O}$. $\square$

This lemma shows that the Cautious Monotonicity (CM) rule is applicable to an axiom with subsumed (lefthand) concept C at rank 0 by left strengthening with the negation of any concept at rank 1. The result can be generalised to concepts with rank greater than 1, but the case considering an axiom at rank 0 and left strengthening concepts at rank 1 is the most interesting because throughout the execution of the suggested methodology, it is only concepts at rank 0 and rank 1 that are considered.

This extension to OntoDebug is visually depicted in Figure 1. The extension is shown in green, while the original OntoDebug methodology is in blue.



Knowledge engineer activities
OntoDebug activities
Start running OntoDebug
No
Inconsistencies/ incoherencies detected?
Yes
Calculate diagnoses
Diagnoses narrowed down sufficiently?
Yes
Satisfied with suggested diag. to repair?
Yes
Accept suggested axiom and make necessary amendments
No
Select axioms to query
Give selected axioms to user as queries
Answer query
Add user answer to positive/ negative test cases
No
No
Click on 'Investigate more'
Isolate the issue
Determine axiom to weaken
Weaken axiom and present result to knowledge engineer
Accept repair?
Yes
Apply solution

Figure 1. OntoDebug extension methodology

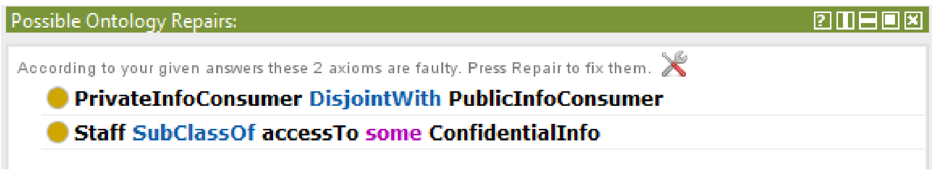
4.1. Applying Axiomatic Weakening to a More Complex Example

To illustrate how the above methodology will be carried out in practice, and work together with the standard interactive ontology debugging framework, consider the following ontology:

$$\mathcal{O} = \left\{ \begin{array}{l} \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{User} \sqsubseteq \exists \mathsf{accessTo.PublicInfo} \\ \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo} \\ \top \sqcap \exists \mathsf{accessTo.PublicInfo} \sqsubseteq \mathsf{PublicInfoConsumer} \\ \top \sqcap \exists \mathsf{accessTo.ConfidentialInfo} \sqsubseteq \mathsf{PrivateInfoConsumer} \\ \mathsf{PrivateInfoConsumer} \sqsubseteq \neg \mathsf{PublicInfoConsumer} \\ \mathsf{ConfidentialInfo} \sqsubseteq \neg \mathsf{PublicInfo} \end{array} \right\}$$

In this example, an entangled inconsistency is present: Staff is an unsatisfiable concept for two reasons: firstly, Staff is unsatisfiable because it is asserted that Staff have accessTo.ConfidentialInfo, yet at the same time, because Staff is subsumed by User, it is also inferred that Staff do not have accessTo.ConfidentialInfo. Secondly, Staff is an unsatisfiable concept because it is inferred that Staff is subsumed by PrivateInfoConsumer because Staff have accessTo.ConfidentialInfo and anything that has accessTo.ConfidentialInfo is considered a PrivateInfoConsumer. Yet, Staff is also subsumed by User, and it is inferred that User is subsumed by PublicInfoConsumer because a User has accessTo.PublicInfo and anything that has accessTo.PublicInfo is considered a PublicInfoConsumer. The incoherence occurs because the concepts PrivateInfoConsumer and PublicInfoConsumer are asserted as being disjoint, yet the concept of Staff has been identified as both a PrivateInfoConsumer and a PublicInfoConsumer.

When following through with the standard OntoDebug methodology, we see that for the above example, two axioms are suggested as in need of repair, due to two minimal conflict sets being involved in causing the incoherence:


Possible Ontology Repairs:

According to your given answers these 2 axioms are faulty. Press Repair to fix them.

● **PrivateInfoConsumer DisjointWith PublicInfoConsumer**

● **Staff SubClassOf accessTo some ConfidentialInfo**

Figure 2. Running entangled concepts through OntoDebug returns separate axioms for repair.

Thus, each repair axiom can be examined individually. Simply removing the axiom asserting disjointness between PrivateInfoConsumer and PublicInfoConsumer solves the first axiom to be repaired – this is also a good example of where it may at times be necessary to simply remove an axiom, rather than attempting to weaken it, as it does make logical sense that someone who is a PublicInfoConsumer can also be a PrivateInfoConsumer.

It does not however make sense to remove or alter the second axiom. In this case, the knowledge engineer can choose to investigate further with the extended debugging methodology to identify relevant axioms that could be weakened rather than deleted. If the knowledge engineer chooses to investigate further in this case, the same steps listed at the beginning of Section 4 will be followed, leading to the result that the recommendation for weakening is $\mathsf{User} \sqcap \neg\mathsf{Staff} \precsim \neg\exists\mathsf{accessTo.ConfidentialInfo}$. Of course, if further levels of exceptionality are present (e.g. the axioms with BlackListedStaff on the LHS), in the next iteration these are picked up on as needing repair, and the same methodology as at the beginning of Section 4 is again followed.

4.2. Axiomatic Weakening as an Ontology Design Pattern

Axiomatic weakening can work both as part of a model-based diagnosis or heuristic approach to ontology debugging. Thus far, we have worked with model-based diagnosis for debugging ontologies. The heuristic approach to debugging tries to find common patterns of faulty ontology modelling and presents suggestions for repairs based on this [21]. The benefit of using the heuristic approach is that, especially with large ontologies, computation of repairs is more efficient as minimal conflict sets do not need to be computed for each inconsistency before returning a result.

Gangemi and Presutti [22] describe an ontology design pattern as a "modelling solution to solve a recurrent ontology design problem". In this case the recurrent ontology design problem is unintuitive exceptionality due to axioms that are stated too strongly. Abstracting away from the User-Staff-BlacklistedStaff example that we have been using up until now, we may define this kind of exception as follows:

Definition 16. *An* ***exceptionality pattern*** *is a recurrent ontology design problem that occurs when, in an ontology $\mathcal{O}$, a concept, H which intuitively must be subsumed by the parent concept, G, causes an inconsistency due to having a relationship r with another concept, I, which is in direct opposition to the relationship that the parent concept G has with the other concept, I.*

That is:

$$\mathcal{O} = \left\{ \begin{array}{l} G \sqsubseteq I \\ H \sqsubseteq G \\ H \sqsubseteq \neg I \end{array} \right\}$$

and intuitively the knowledge engineer would like to still maintain that all of the above axioms are true.

The modelling solution to this recurrent ontology design problem is to weaken the axiom with the most general concept (with the lowest rank) on the left hand side by left-strengthening the most general concept on the left hand side by adding a conjunction with the exceptional concept, as follows: $G \sqcap \neg H \sqsubseteq I$.

5. Conclusion and Future Work

Formal ontologies serve as knowledge representation formalisms over which reasoning tasks can occur. In a vast array of domains, they can be used to formalise knowledge so that axioms are machine-readable and can be reasoned over thus sourcing new knowledge and identifying domain inconsistencies. The success of ontologies thus depends on (1) knowledge retention, (2) without introducing undue logical inconsistencies.

As ontologies are being used in more domains, and especially in domains such as business or legal where there are often exceptions to the rules, the axiomatic intricacy (variety) increases, meaning that inconsistencies *arise more unexpectedly* and evade understanding of how they came about. As inconsistencies arise more often, faster and more frequently evade understanding, the human ability to find adequate solutions for these inconsistencies becomes impaired.

In the same way that Rodler et al. [20,21,11] could motivate the necessity of an interactive ontology debugging methodology by arguing that without it, valuable axioms are often deleted thus leading to a loss of knowledge, our extension can also be motivated: without a strategy showing *how* axioms could be weakened rather than deleted, valuable knowledge may be lost.

For each diagnosis, our extension suggests a way to fix the inconsistency / incoherency by weakening rather than deleting a relevant axiom in the minimal conflict set of that diagnosis. From the point where the knowledge engineer decides to investigate a particular diagnosis returned by OntoDebug in more detail, this is done by:

1. Isolating the issue by pulling through only the selected minimal conflict set (our methodology provides recommendations on which minimal conflict sets would be more apt to address first, though the onus still lies with the knowledge engineer);
2. Determining a candidate axiom to weaken and a candidate concept with which to weaken it by obtaining the ranking of concepts within the minimal conflict set.
3. Weakening the relevant axiom by applying Cautious Monotonicity.

The weakened axioms are returned to the knowledge engineer and they choose to accept or reject the solutions. The full OntoDebug methodology, together with our extension, is followed until all inconsistencies have been resolved, and the ontology is no longer incoherent. We have also shown that axiomatic weakening can be used as an Ontology Design Pattern as part of a heuristic approach to ontology debugging.

We have created a design artifact in the form of a methodology and design plans to suggest how, through the use of defeasible reasoning tools, suggestions of axiomatic weakening could be systematically presented to the user. Our extension enables the usage of a debugging methodology that applies the principle of minimal change in a more nuanced way, thus serving the ultimate goal of knowledge retention in an ontology. This is the main contribution of our work along with the contribution of unearthing rich areas for investigation at the intersection between the defeasible DL and debugging communities.

Future work could focus on using the existing design artifact as a blueprint for an implemented Protégé plug-in, as an extension to OntoDebug. Certain algorithms that play a significant role in the development of this extension have already been implemented: Meyer et al. [23] have, for instance, created the Defeasible Inference Platform (DIP) Protégé plug-in. This plug-in has the ability to rank concepts appearing in defeasible axioms. Furthermore, interactive ontology debugging has been implemented in the OntoDebug Protégé plug-in. Implementation would thus rely on seamlessly merging the existing algorithms, and the design artifact produced by our work will guide the developer in this process. Once implemented, future work could also study the extent to which the effectuated repairs mimic the human non-monotonic reasoning process. Ultimately, this would lead to more robust ontology repairs.

References

[1] Quamar A, Lei C, Miller D, Ozcan F, Kreulen J, Moore RJ, et al. An ontology-based conversation system for knowledge bases. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data; 2020. p. 361-76.

[2] Ziaimatin H, Nili A, Barros A. Reducing consumer uncertainty: Towards an ontology for geospatial user-centric metadata. ISPRS International Journal of Geo-Information. 2020;9(8):488.

[3] Zheng Y, Tetik M, Törmä S, Peltokorpi A, Seppänen O. A Shared Ontology for Logistics Information Management in the Construction Industry. In: ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction. vol. 37. IAARC Publications; 2020. p. 1278-85.

[4] Ismail HO, Shafie M. A Commonsense Theory of Secrets. In: Formal Ontology in Information Systems. IOS Press; 2020. p. 77-91.

[5] Griffo C, Almeida JPA, Guizzardi G. Legal Theories and Judicial Decision-Making: An Ontological Analysis. In: Formal Ontology in Information Systems. IOS Press; 2020. p. 63-76.

[6] Biccheri L, Ferrario R, Porello D. Needs and Intentionality. In: Formal Ontology in Information Systems. IOS Press; 2020. p. 125-39.

[7] Roth D. Learning to reason: The non-monotonic case. In: IJCAI; 1995. p. 1178-84.

[8] Schlobach S, Huang Z, Cornet R, Harmelen F. Debugging incoherent terminologies. Journal of automated reasoning. 2007;39:317-49.

[9] Kalyanpur A, Parsia B, Sirin E, Cuenca-Grau B. Repairing unsatisfiable concepts in OWL ontologies. In: European Semantic Web Conference. Springer; 2006. p. 170-84.

[10] Friedrich G, Shchekotykhin K. A general diagnosis method for ontologies. In: International Semantic Web Conference. Springer; 2005. p. 232-46.

[11] Schekotihin K, Rodler P, Schmid W. OntoDebug: interactive ontology debugging plug-in for Protégé. In: International Symposium on Foundations of Information and Knowledge Systems. Springer; 2018. p. 340-59.

[12] Horridge M, Parsia B, Sattler U. Laconic and Precise Justifications in OWL. In: Sheth A, Staab S, Dean M, Paolucci M, Maynard D, Finin T, et al., editors. The Semantic Web - ISWC 2008. Springer Berlin Heidelberg; 2008. p. 323-38.

[13] McDermott D, Doyle J. Non-Monotonic Logic I. Revision. Massachusetts Institute of Technology – Artificial Intelligence Laboratory; 1979.

[14] Knorr M, Hitzler P, Maier F. Reconciling OWL and non-monotonic rules for the Semantic Web. ECAI 2012. 2012.

[15] Giordano L, Gliozzi V, Olivetti N, Pozzato GL. A non-monotonic description logic for reasoning about typicality. Artificial Intelligence. 2013;195:165-202.

[16] Varzinczak I. A note on a description logic of concept and role typicality for defeasible reasoning over ontologies. Logica Universalis. 2018;12(3-4):297-325.

[17] Britz K, Casini G, Meyer T, Varzinczak I. A KLM perspective on defeasible reasoning for description logics. In: Description Logic, Theory Combination, and All That. vol. 11560 of LNCS. Springer; 2019. p. 147-73.

[18] Britz K, Casini G, Meyer T, Moodley K, Sattler U, Varzinczak I. Principles of KLM-style defeasible description logics. ACM Transactions on Computational Logic (TOCL). 2021;22(1):1-46.

[19] Kraus S, Lehmann D, Magidor M. Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence. 1990;44:167-207.

[20] Rodler P. Interactive Debugging of Knowledge Bases [Ph.D. thesis]. Alpen-Adria University Klagenfurt; 2015.

[21] Rodler P, Jannach D, Schekotihin K, Fleiss P. Are query-based ontology debuggers really helping knowledge engineers? Knowledge-Based Systems. 2019;179:92-107.

[22] Gangemi A, Presutti V. Ontology design patterns. In: Handbook on ontologies. Springer; 2009. p. 221-43.

[23] Casini G, Meyer T, Moodley K, Sattler U, Varzinczak I. Introducing defeasibility into OWL ontologies. In: International Semantic Web Conference. Springer; 2015. p. 409-26.