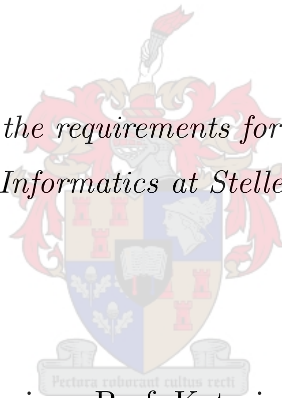
Repairing Classical Ontologies using Defeasible Reasoning Techniques

by

Simone Coetzer



Thesis presented in fulfillment of the requirements for the degree of Master of Arts in the Faculty of Socio-Informatics at Stellenbosch University

Supervisor: Prof. Katarina Britz

March 2021

Stellenbosch University https://scholar.sun.ac.za

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 14 February 2021

Copyright © 2021 Stellenbosch University
All rights reserved.

Stellenbosch University https://scholar.sun.ac.za

Abstract

Ontologies provide knowledge engineers with the ability to represent and encode knowledge in a formal language so that it can be reasoned over by a computer. Notable benefits include the ability to source new knowledge by making statements that are implicitly deduced explicitly available to the end-user, to classify individuals or instances and to check the addition of new knowledge for logical consistency.

Given the nature and goal of ontologies, a successful application of ontologies relies on (1) representing as much accurate and relevant domain knowledge as possible, (2) 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. A myriad of ontology debugging approaches (some of them instantiated in tools) have been developed to help the knowledge engineer pinpoint the cause of logical inconsistencies and rectify them in a strategic way.

Rodler (2015) and Schekotihin et al. (2018) build out the ontology debugging basics by introducing an interactive ontology debugging methodology: this interactive ontology debugging framework, which has also been implemented as a Protégé plug-in, *OntoDebug*, methodically and iteratively asks users queries to narrow down the inconsistency to just one diagnosis, at which time the user can make a more informed decision about how to repair the diagnosis.

This approach guides the user in the debugging process. We show however that this approach can sometimes lead to unintuitive results, which may then lead the knowledge engineer to opt for deleting potentially crucial and nuanced knowledge. This is due to the focus of the interactive ontology debugging approach to be on classical, monotonic knowledge bases – and indeed, in the classical/ monotonic sense, it is only by deletion, not extension of the knowledge base, that coherence can be obtained. However, it may at times be desirable to deal with the unintuitive results produced by *weakening* rather than *deleting* faulty axioms.

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

Stellenbosch University https://scholar.sun.ac.za

(2015) interactive ontology debugging approach which not only localises faulty axioms but provides the knowledge engineer with a strategic way of resolving them by presenting the root cause inconsistencies first. We extend 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 – 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 (as can be found in Britz et al. 2019) 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.

Stellenbosch University https://scholar.sun.ac.za

Opsomming

Ontologieë bied kennisingenieurs die vermoë om kennis in 'n formele taal voor te stel en te kodeer sodat dit deur 'n rekenaar verwerk kan word. Opvallende voordele sluit in die vermoë om nuwe kennis te verkry deur verklarings wat implisiet afgelei is vir die eindverbruiker voor te stel, om individue of gevalle te klassifiseer en om die toevoeging van nuwe kennis na te gaan vir logiese konsekwentheid.

Gegewe die aard en doel van ontologieë, berus 'n suksesvolle toepassing van ontologieë daarop dat (1) soveel akkurate en relevante domeinkennis as moontlik verteenwoordig word, (2) met behoud van logiese konsekwentheid. Aangesien die suksesvolle implementering van 'n industrie-standaard ontologie waarskynlik baie konsepte en ingewikkelde verhoudings tussen die begrippe sal bevat, is dit nodig om 'n metodologie te volg vir die ontfouting en verfyning van die ontologie. 'n Magdom ontologie-ontfoutingsbenaderings (sommige van hulle reeds geïmplementeer) is ontwikkel om die kennisingenieur te help om die oorsaak van logiese teenstrydighede op te spoor en op 'n strategiese manier reg te stel.

Rodler (2015) en Schekotihin et al. (2018) bou die basiese beginsels van ontologie-ontfouting op deur 'n interaktiewe ontologie-ontfoutingsmetodiek in te stel: hierdie interaktiewe ontologie-ontfoutingsraamwerk, wat ook geïmplementeer is as 'n Protégé plug-in, *OntoDebug*, vra die gebruikers iteratiewe en metodiese vrae om die teenstrydigheid tot net een diagnose te beperk, en dan kan die gebruiker 'n meer ingeligte besluit neem oor hoe om die diagnose te herstel.

Hierdie benadering lei die gebruiker in die ontfoutingsproses. Ons toon egter aan dat hierdie benadering soms tot onintuïtiewe resultate kan lei, wat dan kan lei tot die kennisingenieur om potensieel belangrike en genuanseerde kennis te verwyder. Dit is te wyte aan die fokus van die interaktiewe ontologie-ontfoutingsbenadering om op klassieke, monotone kennisbasis te val – en inderdaad, in die klassieke / monotone sin, is dit slegs deur skrapping, nie uitbreiding van die kennisbasis nie, dat samehang verkry kan word. Ons wys egter dat dit selfs in klassieke kennisbasisse soms wenslik kan wees om die foutiewe aksiomas in onintuïtiewe resultate eerder deur *verswakking* as *verwydering* op te los.

Ons bied 'n metodologiese en ontwerpbasis om foutiewe aksiomas op 'n strategiese manier te verswak deur sogenaamde *defeasible* redeneerinstrumente. Ons metodologie put

Stellenbosch University https://scholar.sun.ac.za

uit Rodler se (2015) interaktiewe ontologie-ontfoutingsbenadering wat nie net foutiewe aksiomas lokaliseer nie, maar die kennisingenieur ’n strategiese manier bied om dit op te los deur eers die oorsaak-teenstrydighede aan te bied. Ons brei hierdie benadering uit deur ’n metodologie te skep om stelselmatig aanbevelings oor konflikoplossing te vind. Wat belangrik is, is dat ons doel nie is om ’n klassieke ontologie na ’n defeasible ontologie te omskep nie – daarom gebruik ons nie ’n defeasible redenasie-ondersteuning deur byvoorbeeld die berekening van rasionele afsluiting nie. Ons gebruik eerder die definisie van uitsonderlikheid van ’n begrip, wat sentraal staan in die semantiek van defeasible beskrywingslogika, en die gepaardgaande algoritme (soos gevind in Britz et al. 2019) om die omvang van die konsep se uitsonderlikheid te bepaal (hul rangorde); dan begin ons om verswakte weergawes van die verklarings wat die mees algemene begrippe bevat (die minste uitsonderlike begrippe) voor te stel; dit word gedoen totdat alle teenstrydighede opgelos is.

Stellenbosch University https://scholar.sun.ac.za

Acknowledgements

Firstly, a huge thanks must be extended to Prof. Arina Britz – her inputs have played a pivotal role in shaping raw thoughts into something tangible: she is truly a sculptor of knowledge, and of students' abilities and confidence. I am honoured to have worked with you, and hope that we can work together on future research papers.

Secondly, I'd like to pay thanks to Amanda Viljoen and Jana Neethling. The memorable times we had together make up part of the reason my student years were so fun, and thus acted as a motivation for further study. I'd also like to thank my aunt and uncle Lizelle Brundyn, Koos Brundyn – you have always provided me with humour, and seeing the lighter side of life, which definitely helps especially in times like these.

Bringing a much-needed balance to the lighter side of life, many thanks should also be extended to my brother, Neill Coetzer, the closest to a renaissance man I can think of – always bringing up interesting conversations about art, science and philosophy, and always offering a point of view to go back and think about.

Jev Prentice, my partner in crime (and in life!) – thank you for helping me distance myself from the *"all work and no play"* mantra, and for all of the adventures we have along the way!

Finally, this thesis would not have been possible without the tremendous support from my mom and dad, Karen Brundyn and Wentzel Coetzer. You may not realise it, but both of you are a great inspiration to me, and I cherish the life lessons you have taught me.

You are, each and all of you, *exceptional* in the very best sense of the word.

Stellenbosch University https://scholar.sun.ac.za

Contents

1 Introduction 1
1.1 Research Aims . . . 9
1.2 Thesis Layout . . . 10

2 Background 13
2.1 Ontologies . . . 13
2.2 Description logics . . . 14
2.3 Defeasible DLs . . . 18
2.4 Defeasible Inference Platform in Protégé . . . 26
2.5 Basic ontology debugging principles . . . 29
2.6 Interactive, test-driven ontology debugging . . . 35
2.7 Interactive ontology debugging with OntoDebug in Protégé . . . 40

3 Incorporating Systematic Weakening into Interactive Ontology Debugging 47
3.1 Unintuitive results obtained by interactive ontology debugging . . . 48
3.2 Outline of methodology . . . 52
3.3 Selecting minimal conflict sets and choosing repair axioms . . . 59
3.3.1 Highlighting the most relevant minimal conflict sets: . . . 60
3.3.2 Choosing repair axioms: . . . 62
3.4 Use of systematic weakening as a design pattern . . . 64
3.4.1 Using axiomatic weakening in a heuristic approach to debugging: . . . 64
3.4.2 Using axiomatic weakening in a model-based diagnosis approach to debugging: . . . 67
3.5 Integration with OntoDebug . . . 69
3.5.1 Integration with existing interactive ontology debugging workflow: . . . 69
3.5.2 Integration with initial repair-generation tasks: . . . 72
3.6 Related work . . . 73
3.7 Summary . . . 74

Stellenbosch University https://scholar.sun.ac.za

4 Evaluation and Discussion **76**
4.1 Testing behaviour with edge-cases 76
4.1.1 Entangled inconsistencies/incoherence: 76
4.1.2 Infinity rankings: 78
4.1.3 Context-bound exceptionality: 79
4.2 Contributions to debugging and defeasible DL communities 81

5 Conclusion and Future Work **83**
5.1 Future Work 84

Stellenbosch University https://scholar.sun.ac.za

Chapter 1

Introduction

The benefit of ontologies lies in the fact that they serve as knowledge representation formalisms over which reasoning tasks can occur: by requiring knowledge to be represented, domain experts' implicit knowledge is made explicit; by formalising this knowledge in axioms that are machine-readable, processing can occur over the domain knowledge to source new insights or to warn of inconsistencies (Baader et al. 2004). The success of ontologies thus relies on (1) knowledge retention (so that as much domain knowledge as possible can be accurately preserved) (2) without introducing undue logical inconsistencies.

Ontologies are continually growing in size and complexity. In the same way that the data linked to an ontology's structure is subject to the 3Vs of Big Data (volume, variety, velocity – refer to Banik and Bandyopadhyay (2016)), ontologies too can be argued to now be subject to these 3Vs. Firstly, the volume or size of ontologies is growing. Take for example the popular SNOMED CT ontology which is a collection of medical terms used in clinical documentation and reporting: this ontology currently consists of more than 300 000 concepts and over 1.5 million relations – refer to figure 1 for a visual representation of the scale of concepts and relations within the SNOMED CT ontology.

As ontologies grow in size with multiple human and software agent inputs – as the volume of ontologies increases – it becomes more probable for inconsistencies to arise. Although it may be quite manageable to resolve inconsistencies in small-scale ontologies without the use of any particular approach, a methodology for resolving inconsistencies becomes all the more important with large-scale ontologies as the consequences of 'fixing' an axiom that does not lie at the root cause of the inconsistency may reverberate through the ontology and cause further inconsistencies. For a human to understand the consequences reverberating through the ontology following a fix would be near-impossible and thus it is preferable to follow a formal methodology for pinpointing inconsistencies – and not just pinpointing inconsistencies, but suggesting which ones to solve first (as a root cause inconsistency could be the cause for further inconsistencies further downstream).

Stellenbosch University https://scholar.sun.ac.za

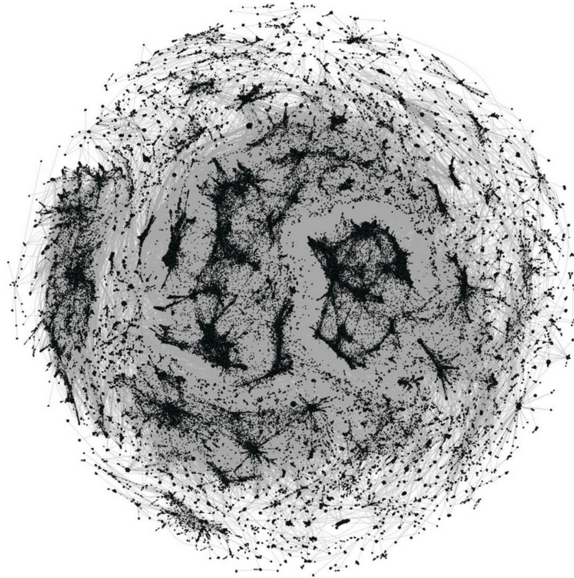


Figure 1 Visualisation of the SNOMED CT ontology. Black nodes represent concepts; grey lines represent relationships between the concepts.

Secondly, the variety or variability of concepts is growing – this is especially the case for ontologies capturing knowledge from the business and legal domains. With concept definitions becoming more complex in nature, is it very likely that in order to solve an inconsistent ontology, axioms would not necessarily need to be deleted, but would rather need to be weakened.

As an example from the domain of business, one of my colleagues expressed his disgrace to me as he was asked for his ID whilst paying for his non-alcoholic beer. If an ontology was involved in product classification, the ontology is likely to have had the following structure:

$$\mathcal{T} = \left\{ \begin{array}{l} \mathsf{Beer} \sqsubseteq \mathsf{AlcoholicBeverage} \\ \mathsf{AlcoholicBeverage} \sqsubseteq \exists \mathsf{requires.Identification} \\ \mathsf{NonAlcoholicBeer} \sqsubseteq \mathsf{Beer} \end{array} \right\}$$

From this ontology, it would be deduced that NonAlcoholicBeer requires.Identification as it is an AlcoholicBeverage. In this case, rather than having the strict definition that Beer is an AlcoholicBeverage we might want to weaken the statement to say Beer is *usually* an AlcoholicBeverage – in a classical ontology, the equivalent of this would be to state that a Beer that is not a NonAlcoholicBeer is an AlcoholicBeverage. This would have enabled us to

Stellenbosch University https://scholar.sun.ac.za

add an axiom that states that NonAlcoholicBeer does not require Identification. As a further example, take the recent explosion of plant-based products – again, certain products like patties or sausages were traditionally classified as meat-based products but now we are forced to concede that they are *usually* meat-based products, thus axioms in product catalogue ontologies would need to be weakened if, for instance, they would like to enable a search for vegetarian products to include vegetarian patties and sausages. With the constant rising trend of using e-commerce over procuring goods from a physical shop, product catalogues would need to ensure the accuracy of online catalogues and buying processes. The point here is that as businesses are evolving and consumer needs are developing, product classification is branching out with the explosion of new products on the market (see for instance Rosnizam et al. (2020) who note that a large part of Tesco's success is due to its variety of products which meet consumer needs; also refer to Tziva et al. (2019) noting the complex product blends emerging especially from the food industry). This often adds complexity (or *variability*) to initial product descriptions. This variability of concept definitions forces traditional ways of classifying concepts in an ontology to be changed more often: i.e. ontology debugging activities take place more often and having tools or techniques to solve these inconsistencies in a manner that captures subtle nuances would decrease the number of hours spent on debugging activities.

As a further case in point, in the legal realm, we see more and more emergent regulatory frameworks like POPI (South Africa's Protection of Personal Information act – inspired by EU's GDPR framework) cropping up. In these cases, it is often suggested that to exercise sufficient control over the risk, ontologies should be utilised (Chan and Hankel 2019). To take an example from this domain: as part of the POPI Act, the following subset of rules are present in Section 72 point 1:

> "A responsible party in the Republic may not transfer personal information about a data subject to a third party who is in a foreign country unless –
>
> 1. the third party who is the recipient of the information is subject to a law, binding corporate rules or binding agreement which provide an adequate level of protection that –
>   - effectively upholds principles for reasonable processing of the information that are substantially similar to the conditions for the lawful processing of personal information relating to a data subject who is a natural person and, where applicable, a juristic person; and

Stellenbosch University https://scholar.sun.ac.za

- includes provisions, that are substantially similar to this section, relating to the further transfer of personal information from the recipient to third parties who are in a foreign country.
- [...]"

Regulator (2013)

From this example, it is clear that legal statements are typically constructed so that the general rule is stated first (in this case this is the first sentence). Then, exceptions to the rule are stated (in this case, this would be the sub-statements). Even from these sub-statements, further sub-statements (or further conditions/ exceptions to the rule) are noted.

The domains of legal and business, as briefly expanded on above, are of course not domains where ontology axioms represent a true, underlying, natural state of the world. They are very much domains built from human interpretation of reality – domains where we do not *discover* the underlying nature of the real world, but rather where we *create* the reality we want to live in. Because of this, is more likely in these domains that certain concepts would need to be more nuanced to allow certain critical axioms to hold whilst having exceptions to the rule. This idea that the knowledge we are uncovering does not reflect the true nature of the world, and that the addition of new knowledge can change our existing beliefs about the world, is in stark contrast with the traditional conception of ontologies where they were thought to represent a model of real world phenomena pointing to the underlying nature of reality (Studer et al. 1998). Despite the philosophical foundations of non-monotonic logics seeming to clash with the initial philosophy of ontologies, it can nonetheless extend the usefulness of ontologies: by developing methodologies for ontologies whereby new knowledge (which is in contradiction to previous knowledge) can be easily added, ontologies are enabled to be more flexible in nature. Furthermore, the nominalist viewpoint can nonetheless resound with another widely-cited definition of ontology – that it is a 'formal, explicit specification of a *shared* conceptualisation' (Studer et al. 1998, p. 184, italics added).

Finally, ontologies are also growing at a faster velocity, especially when one ontology is merged with other ontologies. In industry, this is done to give data analysts and business users easier access to a richer data set. When I first started working on ERP (Enterprise Resource Planning) systems like SAP and Oracle, I was under the impression that these

Stellenbosch University https://scholar.sun.ac.za

types of systems were used to give a full, end-to-end view of business activities in one consolidated data source. Although these systems were developed for this purpose, it is often the case that over time, further extensions to these systems were acquired or built in-house by the business to attain custom functionality (Behrens and Sedera 2004) – this then means that again a fragmented view of the business is obtained and that data from all sources is not readily available thus impeding on business-critical analysis. The aim of ontologies will thus often be to make two or more data sources inter-operable with each other so that a synthesised view of all system data can be obtained (Obrst 2003). To do this, each source system will need an ontology to be developed for them; then the ontologies will need to be merged – this often leads to inconsistencies which would, due to their scale, require a methodological debugging approach or tool to be used to amend axioms to solve inconsistencies.

The above arguments and examples highlight that ontologies are subject to the same 3Vs (volume, variety/variability and velocity) as big data and thus there is a need for a methodical approach to ontology debugging that is also able to cater for the cases where we want the asserted knowledge axioms to be nuanced. To conclude, a methodological approach to ontology debugging is of great use in the following scenarios:

1. **Where the volume of data or size of an ontology is too significant to anticipate the consequences of a manual debugging fix**. We find an example of this in the established bio-medical ontologies like SNOMED CT medical ontology where the number of concepts exceeds 300 000 and the number of relations exceeds 1.5 million.

2. **Where the complexity (also referred to as the variability) of the definitions becomes too nuanced to maintain manually.** Especially in domains like business or legal, where we are not dealing with discovering the underlying nature of reality, but where we are creating our own social reality, it is especially likely that concepts will become more nuanced over time. We can find an example of where this is happening if we consider ontologies for businesses in the retail or manufacturing sector. Where products, their properties and relationships are captured in ontologies, the explosion of product development is likely to lead to new products – products

Stellenbosch University https://scholar.sun.ac.za

that break with traditional classification. This also becomes especially crucial where legal and regulatory frameworks are involved as there are many exceptions that exist to the base rules.

3. **Where the velocity or dynamic nature at which new concepts are added to an ontology increases to such an extent that finding all defects and fixing them in a way so as to retain as much knowledge as possible becomes difficult to achieve manually.** We see an example of this in cases where the ontology of a different system (or in the case of a merger – a different business) is acquired. Along with the general ontology merging steps taken, ontology debugging would also need to be one of the steps taken to resolve merging inconsistencies. Due to the scale of the concepts and relationships present in these ontologies, and due to the addition of concept definition complexity due to the new merger, a systematic ontology debugging methodology would need to be used.

Multiple debugging tools have been developed precisely so that a better methodology for pinpointing the root cause faulty axioms is established (see for instance Schlobach et al. (2007), Kalyanpur et al. (2006) and Friedrich and Schekotykhin (2005)). Notably, the aim of most debugging tools is to pinpoint the diagnoses, or different sets of axioms that could be responsible for leading to the incoherence (refer to Section 2.5 for more detail). Rodler et al. (2019) have suggested however that even though the knowledge engineer is provided with multiple diagnoses based on the input ontology, knowledge engineers often make errors: incorrect diagnoses are selected, and unnecessary deletion of statements ensues. Rodler et al. (2019) and Schekotihin et al. (2018) research how knowledge engineers fare (a) without a debugging tool to guide them; (b) with a debugging tool which provides the diagnoses to the users, but does not interactively lead them through the process of eliminating diagnoses, and (c) with a debugging tool which systematically leads the knowledge engineer through the process of eliminating diagnoses. Firstly, they found that with a debugging tool, the faults were more easily and more quickly found and understood than without a debugging tool. Secondly they found that if the user is not interactively guided through the process of eliminating diagnoses, the incorrect diagnosis is often selected, thus leading to deletion of axioms which contained valuable knowledge. It is on this

Stellenbosch University https://scholar.sun.ac.za

basis that Rodler (2015), Rodler et al. (2019) and Schekotihin et al. (2018) could motivate the use-case for an interactive ontology debugging methodology. 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 – it aims to lead the knowledge engineer to only those statements which *necessarily* require repair. We show however that in the case where what we call 'multi-level exceptions' occur this approach can sometimes lead to unintuitive results (refer to Section 3.1). Furthermore, it does not recommend in which way statements can be weakened in order to be more nuanced. This is due to the focus of the interactive ontology debugging approach to be on classical, monotonic knowledge bases – and indeed, in the classical/ monotonic sense, it is only by deletion, not extension of the knowledge base, that coherence can be obtained.

Drawing from Rodler et al. (2019)'s research, these factors together make it highly likely that a knowledge engineer will opt for *deleting* these seemingly faulty axioms rather than modifying them to encompass knowledge that is more nuanced in nature. This could lead to valuable domain knowledge being lost which would negatively impact on the knowledge retention and thus usefulness of the ontology.

In classical knowledge bases, it may at times be desirable to *weaken* rather than *delete* faulty axioms so that nuanced knowledge is maintained. For example, consider the following axioms in an access-definition ontology, $\mathcal{O}$:

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

In this case, the ontology is incoherent as there is a concept, Staff, which cannot logically have any individuals associated with it. This incoherence arises because an implicit deduction is that Staff do not have access to $\mathsf{ConfidentialInfo}$ (because they are users and a User does not have access to $\mathsf{ConfidentialInfo}$). However an explicit axiom is provided that Staff represent a particular type of User and that this type of User has access to $\mathsf{ConfidentialInfo}$. Although logical incoherence is present according to classical logic, the ontology intuitively makes sense.

Stellenbosch University https://scholar.sun.ac.za

Initially it might be easy for the knowledge engineer to modify the axiom in the following way so that the ontology is made consistent while still aligning with our intuitive expectations:

$$\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubseteq \neg\exists\mathsf{accessTo.ConfidentialInfo}$$

In this case, the knowledge engineer is weakening the initial axiom: this is the classical DL equivalent of weakening the initial axiom with the use of defeasible logic – i.e.: stating that users *usually* do not have access to confidential information (Section 2.3 gives more details on how the notion of exceptionality is used to rank concepts, and consequently to weaken the LHS of the statements). Especially as the levels of exceptionality become more complex, however, the human ability to infer consequences of ontology modification becomes uncoordinated and inelegant. Consider for instance if two more axioms were added to the access-definition ontology, $\mathcal{O}$:

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

This example initially provided by Casini et al. (2013) gives us an idea of the complexity of inconsistency resolution that would be required when multiple levels of exceptionality exist. From the examples related to the 3Vs of Big Data, we know that more and more ontologies are likely to have massive volumes, handle a variety of complex definitions and have concepts added to it at a greater speed. And so, an example like this, especially in an area as sensitive as access control, would need to undergo change via a formal ontology debugging methodology that incorporates, specifically, repair of axioms by systematically presenting weakened versions of axioms to the knowledge engineer (Chapter 3 presents our proposed solution).

It ought to be noted that the automated weakening of the axiom based on a general definition of syntactic or semantic minimal change may also introduce new faulty modelling.

Stellenbosch University https://scholar.sun.ac.za

I.e. it may be the case that an axiom that is flagged as faulty is not just exceptional, but is actually a faulty axiom that ought to be deleted.

From this, it can be argued that this problem can be categorised as a so-called *wicked problem* (Hevner et al. 2004), especially as it involves an IT artifact that has a critical dependence on human interaction to produce accurate solutions, and thus is well-suited to a design science approach. Therefore, we propose a design science artifact in the form of a methodology for systematically generating recommendations given the inconsistency. Although the methodology guides the knowledge engineer in systematically weakening faulty axioms, the onus is still on the knowledge engineer to check whether the recommended solution is truly what they require (refer to Chapter 4 where a discussion and evaluation of our approach is performed). Our approach differs from repair strategies that remove (parts of) axioms, possibly after computing smaller laconic or precise justifications (Horridge et al. 2008). Instead, our methodology aims to identify missing parts of axioms and add them.

1.1. Research Aims

Existing investigations into ontology repair by syntactic weakening have presented important theoretical results (Troquard et al. 2018), but there is no evidence that such results offer practical guidance to ontology engineers in identifying and repairing real ontology design errors. This research is grounded in the Information Systems sub-discipline of design science. Design science views truth not only as something that is *discoverable*, but also as something that is *created* to be of practical use (Hevner et al. 2004, Hevner and Chatterjee 2010). Throughout the course of this research, we will analyse current research in both the debugging and defeasible DL communities thus discovering any gaps that may exist. We then create a design science artifact – a methodology/tool design for obtaining recommendations on rectifying multi-level exceptionality inconsistencies. To provide a methodology/ tool design for inconsistency resolution, two main objectives will need to be met:

1. Defeasible reasoning techniques will be explored to propose how classical axioms leading to multi-level exceptionality inconsistencies can be weakened in a methodical way.

Stellenbosch University https://scholar.sun.ac.za

2. An approach will be developed to explain how the methodical weakening of classical statements (refer to point above) can be incorporated into existing ontology debugging tools and methodologies.

The goal of this research is to provide a methodological foundation for the development and implementation of an inconsistency resolution recommender tool that is fully integrated with an ontology development environment. Importantly, such a methodology/tool is not meant to replace the knowledge engineer's judgement – there may be cases where an axiom simply is redundant and should be deleted and there may also be cases where the knowledge engineer would need to decide whether to switch to a different ontology design methodology.

Once this methodology/ tool design is implemented, the vision is that the tool would enable the knowledge engineer to keep as much relevant domain knowledge as possible by suggesting how problematic axioms may be systematically weakened. The methodology/tool would provide the knowledge engineer with a list of possible fixes showing how problematic axioms have been weakened – the knowledge engineer would then be able to choose which, if any, solution should be picked.

1.2. Thesis Layout

Chapter 2 will provide the reader with the necessary background by introducing them to some foundational concepts necessary to understand the solution provided in Chapter 3. Firstly, this chapter will provide the user with a background on Description Logics (DLs), the logic used to formalise knowledge in an ontology. In this section on DLs, Defeasible DLs will also be expanded upon as in later chapters it becomes imperative to understand the importance of mimicking defeasibility in classical reasoning – it is in this section where it will be explained how axioms can formally undergo weakening so that concepts are defined in a more nuanced way. After this, the second leg of Chapter 2 will provide context and background on core ontology debugging concepts. In this section, we will understand the type of defects which may occur, along with the core steps, across debugging methodologies that are performed to detect defects. Following this basic introduction, existing debugging techniques will be expanded upon, specifically Rodler's (2015) interactive methodology for

Stellenbosch University https://scholar.sun.ac.za

ontology debugging. We show that the advantage of Rodler's (2015) approach is that it does not simply just output all minimal conflict sets at once – rather, it iteratively presents the user with queries, so that the potentially faulty axioms in the diagnosis are pruned away.

After some background has been given on the two most important legs on which this research stands – defeasible description logics and ontology debugging tools – Chapter 3 will then introduce our methodology for obtaining recommendations to resolve multi-level exceptionality inconsistencies. In brief, the following will be illustrated:

1. Firstly, we show that by using the standard interactive ontology debugging methodology, unintuitive results are obtained when multi-level exceptionality inconsistencies exist;
2. We then outline our methodology, which involves using ranking, and defeasible DL postulates – both tools from the Defeasible DL community – to provide a consistent way of weakening identified faulty axioms;
3. As part of our methodology, we then also outline how minimal conflict sets can be selected to apply the weakening algorithm on.
4. The above two points lead us to reflect on formalising axiomatic weakening as a design pattern.
5. Finally we then provide the reader with an end-to-end view of how this extension can be incorporated with the current OntoDebug methodology.

Following a detailed view in Chapter 3 on the proposed extension to OntoDebug, in Chapter 4, an evaluation and discussion of our approach ensues. Specifically, a section is devoted to evaluating the extension's behaviour with edge-cases – these are cases where the inconsistencies/ incoherences are entangled with one another, where a specific concept has a rank of ∞, or where we have context-bound exceptionality. A discussion then shows what contributions this thesis makes to the debugging and defeasible DL communities. The main contribution noted is that the 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. Other spin-off successes include the definition of a design pattern based on weakening rather than deleting axioms; furthermore the work

Stellenbosch University https://scholar.sun.ac.za

opens the floor for further ways in which the Defeasible DL and Debugging communities can lend theory and tools from one another.

Finally, in Chapter 5 our work is summarised and concluded upon, and avenues for future research are listed. This thesis puts forward an extension to interactive ontology debugging that enables, through the use of defeasible reasoning tools, suggestions on how axioms can be repaired by weakening, rather than deleting, the faulty axioms. This work has been done at a design level and in future would need to be implemented as a Protégé plug-in, as an extension to OntoDebug. From there, further user studies can ensue.

Stellenbosch University https://scholar.sun.ac.za

Chapter 2

Background

Two pillars on which the results of this research rest are (1) ontologies, description logics (DLs) and in particular defeasible DLs and (2) ontology debugging tools. This chapter aims to break down these two main areas of focus and to provide a background which would be useful in understanding the topics covered in Chapter 3. In the first half of this chapter, the following is covered: Section 2.1 provides a brief introduction on ontologies as a knowledge representation tool. Section 2.2 provides an overview of Description logics (DLs) as the formal language underlying ontologies. Specifically, this overview provides an introduction on the basic building blocks of DLs, the formation rules of the $\mathcal{ALC}$ language, concept constructors, the semantics of DLs and finally entailment. Section 2.3 provides a brief introductory history on Defeasible DLs; then the defeasible concept constructor is expanded on along with the semantics and entailment of defeasible DLs. Section 2.4 provides an introduction to Protégé, a tool used to model ontologies, and then proceeds to provide a walkthrough of the Defeasible Inference Plugin (DIP) tool used to return a concept's rank given a defeasible knowledge base.

The second half of this chapter, which revolves around ontology debugging, then starts with Section 2.5 to give an overview of the basic ontology debugging concepts and the main steps that are followed. Section 2.6 provides a more detailed look at one specific debugging approach, interactive ontology debugging. This section first provides the benefits of interactive ontology debugging, and then proceeds to give an overview of the main steps involved in this approach. Finally, Section 2.7 provides a walkthrough of the functionality of the OntoDebug tool in Protégé, an interactive debugging tool.

2.1. Ontologies

Knowledge is a crucial resource in ensuring organisational success in contemporary society (Carlile 2002). Specifically in the Knowledge Representation and Reasoning (KRR) com-

Stellenbosch University https://scholar.sun.ac.za

munity, the focus is on formally representing knowledge in the domain of interest so that intelligent applications can use this knowledge (Nardi and Brachman 2003).

There are various ways in which knowledge can be represented – in frames, rule-based systems, taxonomies and ontologies. Ontologies, in particular, are especially attractive because they are deemed to be more reusable and maintainable than rule-based systems. This is mainly because they are created as "explicit specifications of shared conceptualisations" (Studer et al. 1998). Thus they strive to formally capture the underlying nature or "truth" of the real world as it is, independently of an agent's perception of the world (Guarino 1995). Other KR systems like rule-based systems, on the other hand, are designed with a particular functionality in mind. Furthermore, formal ontologies, being reliant on an unambiguous, formal underlying language, provide the groundwork for rich knowledge to be represented computationally – this allows for reasoning activities to be performed over an ontological knowledge base so that new knowledge can be inferred or so that additional knowledge can be checked for consistency (Roussey et al. 2011).

The most well-known framework where OWL ontologies are implemented so that they can be reasoned over and used in practice is in Protégé, a Java-based ontology editor with built-in functionality to allow for concepts to be explicitly described and processed using built-in reasoners. In Protégé ontologies are implemented using OWL (Web Ontology Language) which has its roots in description logics (DLs), discussed in the next section.

2.2. Description logics

A formal language is required to represent knowledge statements (further referred to as 'axioms') unambiguously. Apart from just *representing* axioms, we must also be able to reason over the statements to classify individuals, infer new knowledge or check additional knowledge for consistency. There are multiple formal languages – each formal language has a different expressivity associated with it. The more expressive the language, the more detailed and rich an expression can be; however, with more expressive languages, the computational efficiency decreases – this means that a limit is placed on the reasoning power associated with the ontology. For example, first order logic (FOL), is deemed to be a highly expressive language – it has however been proven to be inherently undecidable

Stellenbosch University https://scholar.sun.ac.za

and so it is not an ideal candidate. On the other hand, less expressive formal logics also exist – these logics would only capture IS-A relationships between concepts and thus rich relationships between concepts cannot be captured. Nardi and Brachman (2003) argue that although these logics are not as expressive, they allow for more complex computational reasoning tasks to be performed over them. Therefore, a harmonious balance needs to be found between the expressivity of a language and its computational power. DLs provide this balance in expressivity – they are fragments of FOL carefully chosen so as to remain decidable, while retaining desirable expressivity of FOL.

Another aspect that separates ontologies from other systems like Relational Database Management Systems (RDMS) is that they operate under the Open-World Assumption (OWA) rather than the Closed-World Assumption (CWA). In the CWA, if information is not stated, then it is assumed that the information is false – in other words, an assumption on completeness of the knowledge base is made. This makes sense for database systems where we expect knowledge to be complete: for instance, if a company owns a purchasing system – you would want to conclude that if an invoice for Dell laptops does not exist, then no Dell laptops were purchased. In contrast to this, with ontologies, which operate under the OWA, if an axiom is not stated or cannot be inferred, then it is simply the case that that axiom is unknown. I.e. it is assumed that the knowledge that is asserted in an ontology is incomplete and therefore only if something is asserted to be false, is that statement false. For example, if we have a purchasing ontology, and it is stated that laptops may be purchased, but nothing is stated on whether cell phones may be purchased, we cannot conclude from this that cell phones may not be purchased – we can only infer that it is unknown whether or not cell phones may be purchased. To use the OWA makes sense in the sphere of ontologies, especially where we assume that we are not dealing with a full knowledge base and that knowledge will, incrementally, be added and extended upon.

A description logic knowledge base consists of two main components – the Tbox and the Abox. The Tbox is where axioms stating the *terminology* of the ontology are stored – it is where subsumption statements and general concept inclusions (GCIs) are formed. For example a statement like 'Mother is a Parent' would be defined in the Tbox. In the Abox, *assertional* statements are provided to define which concept an individual belongs to, or in which relationship an individual is involved in, in relation to another concept. For

Stellenbosch University https://scholar.sun.ac.za

example 'Sally is a mother' is an Abox statement; also the statement 'Sally has a child, Arnold' is an Abox statement where the 'has a' relation is used.

Description logics are referred to in the plural because they are a family of logics, all with different properties. In our research, we focus on the $\mathcal{ALC}$ language. This decision is taken based off of two factors: firstly, $\mathcal{ALC}$ allows for concept negation, and complex concept negation. Especially for the study of inconsistency resolution, it is imperative to use a base language in the study which can cater for negation – the type of inconsistencies we are studying do after all only arise because of the negation of a statement that was previously asserted or inferred to be true.

Secondly, the theoretical foundation that has been created to enable defeasible definitions in description logics has been based off of the $\mathcal{ALC}$ language – thus it makes most sense to build this current research off of this description logic to avoid rework.

Formally, $\mathcal{ALC}$ has the following formation rules:

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

The above statements can be broken down as follows:

1. $\top$ is the top concept used to refer to all individuals within a model.
2. $\bot$ is the bottom concept used to refer to the empty concept (i.e.: the concept with no individuals associated with it);
3. $C \sqcap D$ is the conjunction of C and D. It is used to refer to those individuals belonging to both C and D. This is also referred to as concept intersection. For example, the concept Mother, can be defined as the intersection between the concept Woman and Parent.
4. $C \sqcup D$ is the disjunction of C and D. It is used to refer to those individuals belonging to either C or D. This is also referred to as concept union. For example, a Parent can either be a Mother or a Father.
5. $\exists r.D$ is the existential restriction of r to D. It refers to those individuals that are related by r to some individual in D. For example, $\exists \mathsf{hasChild.Female}$ refers to those individuals who have some $\mathsf{hasChild}$ relationship with some Female individual, i.e. to everyone who has a daughter.
6. $\forall r.D$ is the value restriction of r to D. It refers to those individuals that are related by r to only individuals in D. For example, $\forall \mathsf{hasChild.Male}$ refers to those individuals

Stellenbosch University https://scholar.sun.ac.za

who have a hasChild relationship to only Male individuals, i.e. it refers to everyone who has only sons.

7. $\neg D$ is the negation of D. It refers to the complement of the individuals in D. For example $\neg$Mother refers to those individuals who are not in the concept Mother.

The semantics of a DL language is given by assigning an interpretation, I, which consists of a non-empty set Δ^I (the domain of the interpretation), and an interpretation function which assigns to every atomic concept A a set $A^I \subseteq \Delta^I$ and to every atomic role R a binary relation $R^I \subseteq \Delta^I \times \Delta^I$ (Baader and Nutt 2006). The interpretation function is extended to concept descriptions by inductive definitions for each concept constructor in $\mathcal{ALC}$, e.g.:

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$$

$$\bot^{\mathcal{I}} = \emptyset$$

$$(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$$

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$

$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$

$$(\forall r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in r^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$$

$$(\exists r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in r^{\mathcal{I}}\}.$$

Along with the formation rules, subsumption and concept equivalence are used to define concepts.

1. $\sqsubseteq$ is the subsumption of one concept by another. For example, Mother $\sqsubseteq$ Parent states that Mother is subsumed by Parent. I.e.: Mother is the more specific concept and Parent is the more general concept; Mother is any Parent. Importantly this subsumption does not hold the other way around: the statement does not also by default imply that Parent is any Mother as there are other classes of individuals who are also subsumed by Parent but who are not subsumed by Mother – for example Father.

2. $\equiv$ is the equivalence of one concept to another. For example, Mother $\equiv$ Parent $\sqcap$ Female. With this construct, concepts on either side are equivalent and can be read both ways, i.e.: a Mother is someone who is both a Parent and a Female; similarly, anyone who is a Parent and a Female is a Mother.

A subsumption statement $C \sqsubseteq D$ is true in an interpretation $\mathcal{I}$ if $C^I \subseteq D^I$. $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ if all its elements are true in $\mathcal{I}$. An ABox assertion $C(a)$ is true in an

Stellenbosch University https://scholar.sun.ac.za

interpretation $\mathcal{I}$ if $a^I \in C^I$. An assertion $r(a,b)$ is true in an interpretation $\mathcal{I}$ if $(a,b) \in r^I$. $\mathcal{I}$ is a model of an ontology $\mathcal{O}$ if all its axioms and assertions are true in $\mathcal{I}$.

Definition 1. *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALC}$ ontology, C, D, possibly compound $\mathcal{ALC}$ concepts, and b an individual name. We say that*

1. *C is **satisfiable** with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ and some $d \in \Delta^{\mathcal{I}}$ with $d \in C^{\mathcal{I}}$;*
2. *$\mathcal{O}$ is **consistent** if there exists a model of $\mathcal{O}$*

[(Baader et al. 2017, p. 28)].

Following from the above, an ontology is *inconsistent* if no model can exist for the ontology – i.e. if it contains contradictory facts. A specific concept in an ontology is *unsatisfiable* if it cannot be instantiated without causing an inconsistency. Finally, an ontology is *incoherent* if it contains an unsatisfiable concept. These definitions will be further expanded on, with examples, in Section 2.5.

$\mathcal{ALC}$ has a classical monotonic relation of entailment – this is indicated by $\models$. An ontology $\mathcal{O}$ entails an axiom or assertion α, written $\mathcal{O} \models \alpha$, if α is true in every model of $\mathcal{O}$.

2.3. Defeasible DLs

Classical DLs such as $\mathcal{ALC}$ provide expressivity (which can be further extended with very expressive description logics such as $\mathcal{SROIQ}$), which contributes to the complexity of modelling. As argued in Chapter 1, an industrial need exists to enable ease of debugging in large scale ontologies by returning recommendations on how exceptional concepts could be weakened. A methodological approach to ontology debugging needs to be followed to enable more widespread use of ontologies, especially when ontology merging activities are being performed or when multiple additional concepts need to be added on a regular basis. The reason that the exceptional concepts arise is due to the inability of classical description logics to capture statements in a defeasible form, for example in the form of Mother is *usually* subsumed by the intersection between the concepts Female and Parent. Indeed, even in the social sphere with conceptions around gender evolving (and essentially new knowledge being added to our social ontology) it may now be possible, in exceptional

Stellenbosch University https://scholar.sun.ac.za

cases, for a Male to assume the role of a Mother. Syntactic and semantic extensions of DLs exist which enable us to translate defeasible logic into classical statements which can be processed by monotonic reasoners.

The notion of defeasibility originates from non-monotonic logics. First described by McDermott and Doyle (1980), the notion of non-monotonic logics were formed in contrast to monotonic or classical logics. McDermott and Doyle (1980) 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. Indeed, with the examples given up to this point (the SNOMED example that shows an ontology growing in size, the access example showing that business rules become more nuanced, the systems-merge example), it is clear that knowledge is evolutionary in nature. Therefore, rather than uncovering the true structure of the world around us, we are building a Quinean 'web of belief' where, with each new addition of knowledge, the web adjusts itself to reflect the new structure.

Monotonic logics have the property of 'extension' meaning that the theorems of a set of axioms are always a subset of the theorems of any extension of the set of axioms – in other words, that whatever new knowledge is added would not invalidate old axioms (McDermott and Doyle 1980, p. 5). Initially, non-monotonic logics were only defined in the negative, i.e.: as *not* having the property of extension. Only later, Shoham (1987) and Kraus et al. (1990) were able to define non-monotonic logics positively with the use of preferential models on propositional logic – this is referred to as the *KLM approach*.

Several non-monotonic extensions of DLs exist (Knorr et al. (2012); Giordano et al. (2013); Varzinczak (2018)). Britz et al. (2019, 2020) extended the work of Kraus et al. (1990) 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 using this approach on DLs. 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

Stellenbosch University https://scholar.sun.ac.za

tasks. Defeasible subsumption, also referred to as Defeasible Concept Inclusion, is defined in the following way:

Definition 2. *Let $C, D \in \mathcal{L}$. A **defeasible concept inclusion** axiom (DCI for short) is a statement in the form $C \sqsubsetsim D$ [Britz et al. (2019)].*

Statements that are written in the form $C \sqsubsetsim D$ should be read as 'C is *usually* subsumed by D' or 'individuals that are typical C's are also elements of D'. The symbol denoting defeasible subsumption $\sqsubsetsim$ can thus be used in the same way as a normal subsumption $\sqsubseteq$ is used with the difference being that it refers to defeasible concept inclusion.

To continue with an example from earlier, we may rewrite some of the statements into their defeasible forms in a defeasible T-box, in knowledge base $\mathcal{O}$ so that the ontology is now coherent:

Definition 3. *A **defeasible T-box** (dTbox for short) is a finite set of DCIs and GCIs [Britz et al. (2019)].*

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

The first and third statements now read, respectively: a User *usually/normally* does not have access to $\mathsf{ConfidentialInfo}$ and Staff *usually/normally* have access to $\mathsf{ConfidentialInfo}$. From this then, it is clear that $\sqsubsetsim$ is the defeasible counterpart of $\sqsubseteq$ as it also acts as a connective positioned between the concept language (the object level) and the meta-language (the level of entailment). The semantics of $\sqsubsetsim$ is defined w.r.t. preferential interpretations.

Definition 4. *A **preferential interpretation** is a structure $\mathcal{P} := \langle \Delta^{\mathcal{P}}, \cdot^{\mathcal{P}}, \prec^{\mathcal{P}} \rangle$ where $\langle \Delta^{\mathcal{P}}, \cdot^{\mathcal{P}} \rangle$ is a DL interpretation (which we denote by $\mathcal{I}_{\mathcal{P}}$ and refer to as the classical interpretation associated with $\mathcal{P}$), and $\prec^{\mathcal{P}}$ is a strict partial order on $\Delta^{\mathcal{P}}$ (i.e., $\prec^{\mathcal{P}}$ is irreflexive*

Stellenbosch University https://scholar.sun.ac.za

and transitive) satisfying the smoothness condition (for every $C \in \mathcal{L}$, if $C^{\mathcal{P}} \neq \emptyset$, then min $_{\prec^{\mathcal{P}}}\left(C^{\mathcal{P}}\right) \neq \emptyset$) [Britz et al. (2019)].

Definition 5. *A defeasible subsumption relation $\sqsubsetsim$ is a* ***preferential subsumption relation*** *if it satisfies the following set of properties, called the preferential KLM properties for DLs [Britz et al. (2019)]:*

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

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

Along with the above properties, if the relation $\sqsubsetsim$ also satisfies rational monotonicity (RM), then it is a *rational* subsumption relation (Britz et al. 2019):

$$(\text{RM})\ \frac{C \sqsubsetsim D,\ C \not\sqsubsetsim \neg E}{C \sqcap E \sqsubsetsim D}$$

Definition 6. *Given $C, D \in \mathcal{L}$, a statement of the form $C \sqsubsetsim D$ is a defeasible subsumption statement. A preferential interpretation $\mathcal{P} = \left\langle \Delta^{\mathcal{P}}, \cdot^{\mathcal{P}}, \prec \mathcal{P} \right\rangle$* ***satisfies*** *a defeasible subsumption statement $C \sqsubsetsim D$, if $min_{\prec \mathcal{P}}\left(C^{\mathcal{P}}\right) \subseteq D^{\mathcal{P}}$ [Britz et al. (2019)].*

It is necessary 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 crucial to the concept of 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. McCarthy urges the reader to see the preferential models as different 'worlds' that the reasoner uses – if an exception is encountered, the exception is simply seen as less normal, or less preferred, 'world' in which reasoning is performed (in Kraus et al. (1990)). An object's normality or typicality is determined not by some intrinsic characteristic that the object possesses, but rather it is determined in relation to

Stellenbosch University https://scholar.sun.ac.za

the other objects in the domain. This is referred to as the modular order of an object. The assumption of rationality (RM) imposes a further restriction on preference orders, namely that they are modular. This partitions the domain into layers which are linearly ordered.

Definition 7. *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 [Britz et al. (2019)].*

Definition 8. *A* ***modular interpretation*** *is a preferential interpretation $\mathcal{R} = \langle \Delta^{\mathcal{R}}, \cdot^{\mathcal{R}}, \prec \mathcal{R} \rangle$ such that $\prec^{\mathcal{R}}$ is modular [Britz et al. (2019)].*

In the same way as classical DL interpretations are constrained by a classical DL knowledge base to those subjectively deemed to be consistent with reality, preference orders are also, in the words of Boutilier, 'purely subjective' (Boutilier 1994) – that is to say that by following this methodology of ranking objects, we can "encode our *expectations* about the objects corresponding to our perceived regularity or typicality" (Britz et al. 2017, p. 5). Furthermore, these rankings may also be constrained by empirical data (Britz et al. 2017, p. 5). Whereas classical ontologies then usually follow the philosophical 'rationalist' way of thinking about obtaining knowledge, this addition of subjectively ranking objects according to their typicality bridges the rationalist roots of ontologies with empiricism by imposing a preferential interpretation on objects.

It is evident that this represents a monotonic entailment relation, which thus reduces defeasible reasoning to classical reasoning, and modular closure is not necessarily rational.

Definition 9. *A statement α is* ***modularly entailed*** *by a defeasible knowledge base $\mathcal{O}$, written $\mathcal{O} \models_{mod} \alpha$ if every modular model of $\mathcal{O}$ satisfies α [Britz et al. (2019)].*

Importantly, the notion of exceptionality is central to computing a concept's rank – the more exceptional a concept is, the higher its rank is:

Definition 10. *Let O be a defeasible knowledge base and $C \in L$. We say C is* ***exceptional*** *in $\mathcal{O}$ if $\mathcal{O} \models_{mod} \top \sqsubsim \neg C$. A DCI $C \sqsubsim D$ is exceptional in $\mathcal{O}$ if C is exceptional in $\mathcal{O}$ [Britz et al. (2019)].*

Stellenbosch University https://scholar.sun.ac.za

To understand how the ranking algorithm works in practice, let's continue to work with the access example from earlier:

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

Rather than immediately noting the set of the above axioms in red as incoherent, a ranking algorithm can be applied to rank statements according to their 'typicality' or 'normality' based on how exceptional they are. The ranking algorithm works as follows (Britz et al. 2019):

1. First, the left-hand-side concept of all defeasible statements that are non-exceptional (according to Definition 10) are given a ranking of 0. The DCIs with non-exceptional left-hand side concepts are also given 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. As before, the DCIs with a non-exceptional left-hand side concept are also given 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.
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.

If the above steps were applied to our example, the following would be the outcome:

1. **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.

   In accordance with Definition 10, the concept User is not exceptional w.r.t. $\mathcal{O}_0$, and so the following statement is also not exceptional.

Stellenbosch University https://scholar.sun.ac.za

$$\mathcal{O}_0 = \left\{ \mathsf{User} \mathrel{\underset{\sim}{\sqsubset}} \neg\exists \mathsf{accessTo.ConfidentialInfo} \right\}$$

2. **Ranking of 1 – concepts that are exceptional w.r.t. level 0 statements.** Then, a new knowledge base is created containing only the classical statements, together with the remaining exceptional statements after the statements in $\mathcal{O}_0$ have been removed.

   The new knowledge base would thus contain the following statements:

$$\mathcal{O} = \left\{ \begin{array}{l} \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \mathrel{\underset{\sim}{\sqsubset}} \exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{BlackListedStaff} \sqsubseteq \mathsf{Staff} \\ \mathsf{BlackListedStaff} \mathrel{\underset{\sim}{\sqsubset}} \neg\exists \mathsf{accessTo.ConfidentialInfo} \end{array} \right\}$$

   The following statements in the new knowledge base are now deemed to be 'most normal' in this context:

$$\mathcal{O}_1 = \left\{ \mathsf{Staff} \mathrel{\underset{\sim}{\sqsubset}} \exists \mathsf{accessTo.ConfidentialInfo} \right\}$$

   Therefore, the concept Staff now has a ranking of 1.

3. **Ranking of more than 1 – concepts that are exceptional to other exceptions.** The above step is iterated until all axioms have been assessed. In this case, again a new knowledge base is created using only the remaining exceptional statements together with the classical statements.

$$\mathcal{O} = \left\{ \begin{array}{l} \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{BlackListedStaff} \sqsubseteq \mathsf{Staff} \\ \mathsf{BlackListedStaff} \mathrel{\underset{\sim}{\sqsubset}} \neg\exists \mathsf{accessTo.ConfidentialInfo} \end{array} \right\}$$

   In the context of this new knowledge base, the most normal concept is now BlacklistedStaff and as the rank is increased by 1, this concept now gets a rank of 2.

$$\mathcal{O}_2 = \left\{ \mathsf{BlackListedStaff} \mathrel{\underset{\sim}{\sqsubset}} \exists\neg \mathsf{accessTo.ConfidentialInfo} \right\}$$

4. **Ranking of ∞ – concepts that are unsatisfiable in all ranks.** Once the algorithm does not reveal any new non-exceptional concepts of any ranking, if there are any

Stellenbosch University https://scholar.sun.ac.za

concepts that remain exceptional they are given a rank of ∞. In our case, there are no further concepts. An example of what would result in a concept with an infinite ranking is if we, either explicitly or implicitly stated that 'BlackListedStaff have access to confidential information' – then, not only would the concept of BlackListedStaff be exceptional when taken together with the other statements involving its parent concepts, User and Staff, but it would be exceptional in relation to itself because it is being asserted that they both have and do not have access to confidential information, and this assertion is being made on the same concept level itself, not on the level of a concept higher up in the hierarchy.

In summary, the following results would be obtained after the ranking formula is applied:

World order/ rank	Concept
0	User
1	Staff
2	BlackListedStaff

Table 1 Example of ranking output.

In this table, the concept associated with a rank of 0 is the most normal/ most typical concept – these can be seen as the concepts that could be instantiated even if all the axioms were classical. The concepts associated with rank 1 are those that could only be instantiated if there were at least 1 level of exceptionality. Finally, those with ranking 2 require at least 2 levels of exceptionality.

Semantically, classical subsumption statements of the form $C \sqsubseteq D$ can be rewritten to defeasible statements:

Lemma 1. *For every preferential interpretation, $\mathcal{P}$, and every $C, D \in \mathcal{L}, \mathcal{P} \models C \sqsubseteq D$ if and only if $\mathcal{P} \models C \sqcap \neg D \sqsubseteq \bot$ [Britz et al. (2019)].*

To understand how defeasible statements would be consumed by a classical reasoner, it is also necessary to explore the notion of rational entailment. Rational closure provides a proof-theoretic characterisation of rational entailment. Rational closure is an inferential closure based on modular entailment, but it extends its inferential power.

Definition 11. *Let $\mathcal{O}$ be a defeasible knowledge base and $C, D \in \mathcal{L}$.*

Stellenbosch University https://scholar.sun.ac.za

1. *$C \precsim D$ is in the* ***rational closure*** *of $\mathcal{O}$ if*
   $rank_{\mathcal{O}}(C \sqcap D) < rank_{\mathcal{O}}(C \sqcap \neg D)$ *or* $rank_{\mathcal{O}} C = \infty$.
2. *$C \sqsubseteq D$ is in the rational closure of $\mathcal{O}$ if* $rank_{\mathcal{O}}(C \sqcap \neg D) = \infty$

[Britz et al. (2019)].

In a more colloquial manner, the above states that if an axiom is a defeasible axiom, the defeasible axiom is in the rational closure of the knowledge base if, from assessing the ranking of the statements, it is the case that C typically leads to D and only in exceptional cases does C not lead to D. Classical subsumption statements are in the rational closure of the knowledge base if it is never the case that C is not subsumed by D.

Although our work will not focus on converting classical statements into defeasible statements, it will draw from some of the concepts mentioned in the above section. Specifically in Chapter 3 we will refer to the ranking algorithm as this will help to systematically determine which inconsistencies to solve. Furthermore, we will also be referring to the postulates noted in Definition 5, as these give us a way of rewriting classical statements into their weakened form.

2.4. Defeasible Inference Platform in Protégé

The above theory has been implemented as a Protégé 5.0 plug-in by Meyer et al. (2014) as the Defeasible Inference Platform (DIP) plug-in. The following sets out the purpose of the plug-in and will briefly outline the functionality and user interface of the plug-in. This is done as in Chapter 3, along with the foundational theory, we will provide an understanding of the desired flow by building on this tool, and the OntoDebug tool discussed in Section 2.7.

For ontologies to be implemented in practice, the Web Ontology Language OWL is used. OWL 2 DL is a family of fragments of OWL 2 corresponding to description logics, and thus OWL 2 languages are also characterised by their use of formal semantics. In OWL, ontologies are represented in RDF (Resource Description Framework) which is the XML standard agreed upon by the World Wide Web consortium (W3C). The most popular ontology editor is Protégé, the latest version (as at the date of writing) being version 5.5 (Aminu et al. 2020) [1].

[1] https://Protege 5.0.stanford.edu/

Stellenbosch University https://scholar.sun.ac.za

In Protégé 5.0 we implemented the same ontology, $\mathcal{O}$, as in previous examples. First, the atomic class hierarchy was created – in this case, we have asserted that BlackListedStaff is a subclass of Staff and that Staff is a subclass of User. The concept ConfidentialInfo was also created.

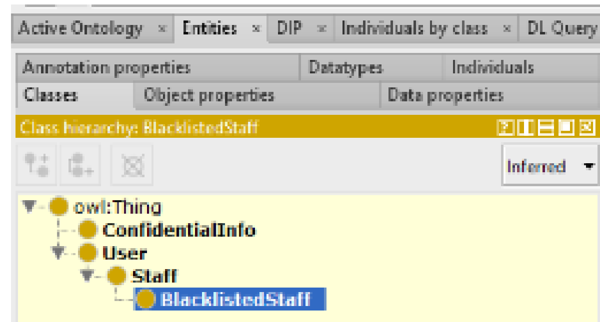


Figure 2 Example of class hierarchy in Protégé.

Then, further subsumption statements were added for each relevant concept. To do this, the role accessTo was first created in the object properties tab in Protégé 5.0. See below for the subsumption statements that were created for User, Staff and BlackListedStaff. Note that the keyword 'some' in Protégé denotes an existential restriction.

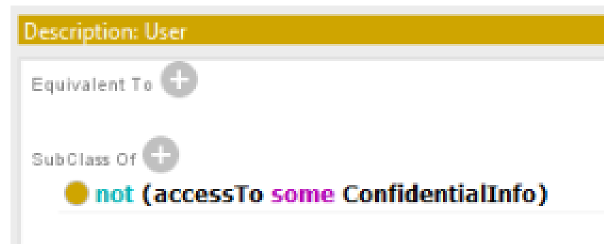


Figure 3 Subsumption statement associated with User.

Also note that the axioms that are explicitly added appear in the 'SubClass Of' box; axioms that are inferred appear in the 'SubClass Of (Anonymous Ancestor)' box – from a classical perspective, it is already evident in the Staff and BlacklistedStaff boxes that these axioms would lead to an incoherence.

Stellenbosch University https://scholar.sun.ac.za

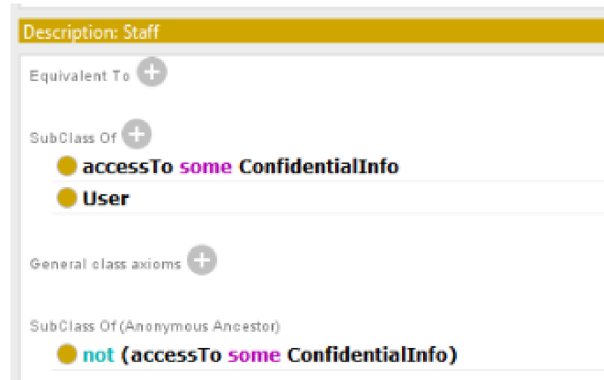


Figure 4 Subsumption statement associated with Staff.

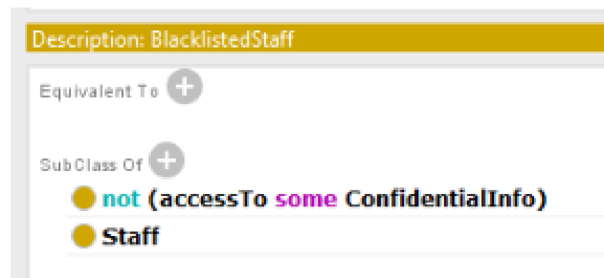


Figure 5 Subsumption statement associated with BlackListedStaff.

Finally, in the Defeasible Inference Plugin (DIP) tab in Protégé, we can select the statements that we would like to keep as classical axioms or make defeasible – this is done by toggling the 'd' button to the right of the axiom. Note that all axioms can be toggled as defeasible if a user is unsure which axioms are defeasible and the user would just like to understand how the axioms rank one against the other. Importantly, if a concept does not occur in any statement marked as defeasible, it will not appear in the ranking.

Figure 6 Axioms noted to be strict/ classical axioms.

When the 'refresh' button is clicked at the very bottom of the tab, the concepts are ranked – we can see that the tool deduced that Staff has a ranking of 1 thus being more

Stellenbosch University https://scholar.sun.ac.za

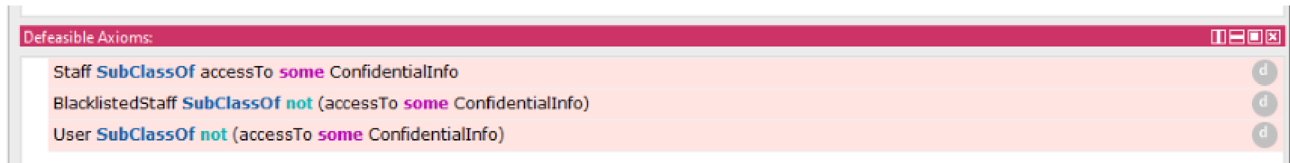


Figure 7 Axioms noted to be defeasible axioms.

exceptional than concepts on level 0 (not shown as these concepts are on the classical level) and that BlacklistedStaff has a ranking of 2 thus being more exceptional than Staff which is on level 1.

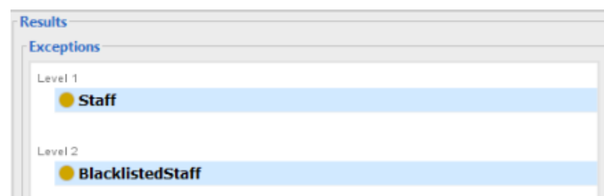


Figure 8 Ranking in DIP in Protégé 5.0.

From this, and the preceding section, the reader should now have a good understanding of the semantics of defeasible subsumption. In Chapter 3, the definitions used in the previous section will be used to explain how interactive ontology debugging methodologies can be extended and return more nuanced repairs by calling upon tools used in defeasible semantics. In Section 3.5, the functionality of the DIP tool discussed in this chapter will be referred to so that the practical usage of the suggested methodology becomes more clear.

The next three sections will follow a similar structure, whereby first the theoretical aspects relating to debugging are explored in Sections 2.5 and 2.6. Then, an overview of the functionality of the OntoDebug tool in Protégé will be given.

2.5. Basic ontology debugging principles

In Chapter 1, the need for formal ontology debugging methodologies/tools was expressed by arguing that ontologies are subject to the same 3Vs that Big Data is subject to: volume (size), variety (variability or complexity) and velocity. A number of well-known ontologies are large in size – concepts and relationships in a number of ontologies are already

Stellenbosch University https://scholar.sun.ac.za

hundreds of thousands with each concept being added giving rise to multiple additional relationships. The variety (or complexity) encountered in ontology definitions is likely to increase as concepts become more nuanced either because new knowledge is uncovered about the concept (as would occur in the medical domain) or because new concepts are developed (as is often the case in the business domain as innovation ensues). Finally, the velocity of concept and relationship additions in an ontology can be high, with multiple users and potentially even systems contributing to ontology development. The velocity at which concepts and relationships are added becomes especially pronounced when ontologies are merged (as can happen when business mergers occur and systems between two companies need to align). As discussed in Chapter 1, these factors require that there needs to be a formal methodology (preferably instantiated in a tool) to deal with exceptions systematically as the human ability to infer consequences that reverberate through the system is limited.

In this section then, we will firstly lay down the basic principles associated with ontology debugging – these concepts are the basic concepts present in further nuanced ontology debugging approaches. Then, after these basic concepts are laid down, we will explore specifically Rodler's (2015) interactive ontology debugging tool.

First, it is important to understand different overarching approaches to debugging ontologies. The main approach used is model-based diagnosis: the observed state of the ontology is compared to the desired state (Schekotihin et al. 2018, p. 16). Generally, the desired state is to have a consistent, coherent ontology. If the current state of an ontology is not consistent or coherent, then strategies are employed to identify what is causing the inconsistency or incoherence and how these can be addressed. The limitation with this approach is that only if an inconsistency or incoherence occurs will the user be aware of it – there are however some cases where a modelled ontology is not inconsistent or incoherent, but nonetheless incorrect inferences are drawn. An example of this is where historically from the SNOMED ontology it was inferred that every amputation of a finger was an amputation of a hand (Peñaloza 2019) – this is clearly an unintuitive and incorrect result, even though no logical incoherence is present. Although these faults will not be picked up on when first running an ontology debugging tool based on model-based diagnosis, it is possible to flag this inference as an unwanted inference once the fault is found – more in this in Section 2.7. A way that these faults can be proactively picked up on is via heuristic approaches

Stellenbosch University https://scholar.sun.ac.za

(refer to Rodler and Schmid (2018), Yamada and Fukuta (2016), Wang et al. (2005)) – examples of heuristic approaches include finding design patterns that are often associated with ontology faultiness, or even learning from the user's previous faults to detect whether another fault has been potentially made. For the purposes of this research, the main focus is on a model-based diagnosis approach.

Within the context of model-based diagnosis, we can identify two types of defects (Lambrix 2019): syntactic defects are defects caused by incorrect formation of the description logic language – these are usually picked up by a parser and are relatively easy to update. As these are generally not caused by ill-formed logic, they will not have consequences that reverberate through an ontology and thus our focus is not on these kinds of defects. Then there are semantic defects which this research will focus on – these are generally thought to have more severe impacts than syntactic defects and are generally more difficult to solve and if solved incorrectly, will lead to other downstream logical problems, or will lead to incorrect query results being returned.

There are three semantic defects which could occur: inconsistency, unsatisfiability and incoherence (Soylu et al. 2013). An ontology is *inconsistent* if no model can exist for the ontology – i.e. if it contains contradictory facts (refer to Definition 1). A specific concept in an ontology is *unsatisfiable* if it cannot be instantiated without causing an inconsistency (refer to Definition 1). Finally, an ontology is *incoherent* if it contains an unsatisfiable concept (refer to Section 2.2).

Drawing from the user access example introduced earlier, we can say that the concept Staff would have been unsatisfiable if, in addition to the Tbox axiom, we had an Abox axiom where it was asserted that an individual, Gloria, is an element of the concept Staff – thus the entire ontology would be inconsistent. This is due to Staff being subsumed by User, so they **do not** have access to ConfidentialInfo, yet contradictory to this it is also stated that they **do** have access to ConfidentialInfo.

$$\mathcal{T}_{\mathcal{O}} = \left\{ \begin{array}{l} \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo} \end{array} \right\}$$

$$\mathcal{A}_{\mathcal{O}} = \left\{ \mathsf{Staff(Gloria)} \right\}$$

Stellenbosch University https://scholar.sun.ac.za

Using the same example, we can note that even before adding the Abox axiom which made the ontology as a whole inconsistent, the concept is unsatisfiable because it is not logically possible for any individual to belong to a concept where they both do not have access to ConfidentialInfo and have access to ConfidentialInfo. This means that before even adding the Abox axiom which made the ontology inconsistent, the ontology as a whole was incoherent because it contains an unsatisfiable concept, Staff.

We now turn to the main steps involved in ontology debugging: detecting defects and repairing defects (Lambrix 2019). To detect defects, the justifications leading to the inconsistency need to be found.

Definition 12. *Let $\mathcal{T}$ be a Tbox and $\mathcal{T} \models \psi$. A set of axioms $\mathcal{T}' \subseteq \mathcal{T}$ is a* ***justification*** *for ψ in $\mathcal{T}$ if $\mathcal{T}' \models \psi$ and $\forall \mathcal{T}'' \subsetneq \mathcal{T}' : \mathcal{T}'' \not\models \psi$ [(Lambrix 2019)].*

Rodler (2015) notes that whereas the term 'justification' is usually used by the general KRR community, the term does not necessarily have any direct link with only ontology *defects*. That is, the term is mostly used to refer to finding a subset of axioms in an ontology that leads to a specific conclusion – in the context it is used, it is mostly used to refer to those statements which lead to a specific *valid* conclusion. For this reason, in the diagnosis community, justifications for defects are referred to as conflict sets (also referred to as Minimal Unsatisfiability Preserving Sub-Tboxes – MUPS).

Definition 13. *Let $\mathcal{T}$ be a Tbox and P be an unsatisfiable concept in $\mathcal{T}$. A set of axioms $\mathcal{T}' \subseteq \mathcal{T}$ is a* ***conflict set*** *if P is unsatisfiable in $\mathcal{T}'$*

The conflict set $\mathcal{T}$ is ***minimal*** *if and only if there is no conflict set $\mathcal{T}'$ where $\mathcal{T}' \subset \mathcal{T}$ [(Lambrix 2019) and (Schekotihin et al. 2018)].*

A minimal conflict set is obtained to avoid computing justifications for *purely derived* causes. For instance, it may be the case that one concept, B, is purely unsatisfiable because it is a subset of another concept A, that is unsatisfiable. We would want to return the set of axioms leading to the inconsistency in A first, as when the axioms leading to the defect in A is solved, B may automatically be solved.

To illustrate these definitions with an example, let's consider an extended version of the access example we have been working with up until this point:

Stellenbosch University https://scholar.sun.ac.za

$$\mathcal{T}_{\mathcal{O}} = \left\{ \begin{array}{l} \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{User} \sqsubseteq \exists \mathsf{accessTo.GeneralInfo} \\ \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.GeneralInfo} \\ \mathsf{Director} \sqsubseteq \mathsf{Staff} \\ \end{array} \right\}$$

$$\mathcal{A}_{\mathcal{O}} = \left\{ \begin{array}{l} \mathsf{Staff(Gloria)} \\ \mathsf{User(Peter)} \\ \mathsf{Director(Susan)} \end{array} \right\}$$

In this example, it is evident that it is not all the axioms that are leading to an inconsistency – only a subset of axioms. In this case, the conflict set that leads to the entailment that the concept of Staff is unsatisfiable and the conflict set that leads to the entailment that the concept of Director is unsatisfiable consists of the following subset of statements, respectively:

That Staff is unsatisfiable:

$$\mathcal{T}_{\mathcal{O}} = \left\{ \begin{array}{l} \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo} \end{array} \right\}$$

$$\mathcal{A}_{\mathcal{O}} = \left\{ \mathsf{Staff(Gloria)} \right\}$$

Stellenbosch University https://scholar.sun.ac.za

That Director is unsatisfiable:

$$\mathcal{T}_{\mathcal{O}} = \left\{ \begin{array}{l} \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Director} \sqsubseteq \mathsf{Staff} \end{array} \right\}$$

$$\mathcal{A}_{\mathcal{O}} = \left\{ \mathsf{Director(Susan)} \right\}$$

From the above, it is evident that the only reason Director is part of the conflict set is because it is subsumed by Staff which is at this stage an unsatisfiable concept. Therefore, the minimal conflict set is the first three axioms above.

For ontology defect detection, we have now seen the main concepts at play. The second step in ontology debugging is ontology repair. In the literature, there are definitions for what would be deemed as a repair (something which, after making the repairs – usually through deletion – the inconsistencies are resolved):

Definition 14. *Let $\mathcal{T}$ be a Tbox and C be the set of all atomic concepts in $\mathcal{T}$. Let M be a finite set of missing axioms. Let W be a finite set of incorrect Tbox axioms. Let Or be an oracle that given a Tbox axiom returns true or false. A* ***repair*** *for Complete-Debug-Problem $CDP(T, C, Or, M, W)$ is any pair of finite sets of Tbox axioms (A, D) such that*

1. $\forall p \in A : Or(p) = true;$
2. $\forall q \in D : Or(q) = false;$
3. $(T \cup A) \setminus D$ *is consistent;*
4. $\forall m \in M : (T \cup A) \setminus D \models m;$
5. $\forall w \in W : (T \cup A) \setminus D \not\models w$

[(Lambrix 2019)].

Although there is a definition for what constitutes a repair and even (as we shall see in the next section) pinpointing which axioms need repair, as stated in Chapter 1, in general there are no methodologies, algorithms or tools that provide a recommendation on how axioms can be systematically weakened to repair ontologies. There are however currently

Stellenbosch University https://scholar.sun.ac.za

ways to assess a manually implemented repair based on how much it changes the original ontology: one way to measure a repair is to determine the extent of change the repair brings into an ontology, with minimal change being sought after as this would preserve the completeness of the ontology. In Chapter 3 we provide a methodology for weakening, rather than deleting, faulty axioms.

2.6. Interactive, test-driven ontology debugging

The basic ontology debugging concepts have now been covered. 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) which represent the root cause of the problem. Consider the following example faulty ontology, $\mathcal{O}$:

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

$$\mathcal{A}_{\mathcal{O}} = \left\{ \mathsf{Staff(Maggie)} \right\}$$

In the above, all axioms except the final Tbox axiom are considered to be part of the minimal conflict set. The reason for this is as follows: Because a User has access to $\mathsf{PublicInformation}$, a user is considered to be a $\mathsf{PublicInfoConsumer}$. Because Staff have access to $\mathsf{PrivateInformation}$ they are considered to be $\mathsf{PrivateInfoConsumers}$. It is also an inferred consequence that Staff are $\mathsf{PublicInfoConsumers}$ because Staff are Users. However,

Stellenbosch University https://scholar.sun.ac.za

it is asserted that a PrivateInfoConsumer is not a PublicInfoConsumer, and because Staff is explicitly stated to be a PrivateInfoConsumer whilst being a PublicInfoConsumer, the concept of Staff is unsatsifiable. Because this unsatisfiable concept is instantiated by Maggie, the ontology is inconsistent.

The logic leading to this type of inconsistency is a mistake that can be made easily. Especially because it logically makes sense that PrivateInformation is disjoint from PublicInformation, it may be easy to intuitively assert that therefore PrivateInfoConsumer is disjoint from PublicInfoConsumer. From manual inspection, it is in fact only the second last Tbox axiom that needs to be removed in order to effectuate a repair. Rodler (2015) notes that generally, humans – even those with knowledge of the representation language being worked with – find it hard to formulate correct logical axioms, or to understand what is causing the problem. In user studies done on groups who were using Rodler's OntoDebug tool, subjects commented that without the guidance of the tool, they would have found it very difficult to understand the nature of the incoherence/ inconsistency, even with the minimal conflict set being returned (Rodler et al. 2019).

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 (Schekotihin et al. 2018, p. 6-7).

Definition 15. *Let $\mathcal{O}$ be a knowledge base, and let $\mathcal{B} \subseteq \mathcal{O}$ be the* background knowledge *to $\mathcal{O}$. Then all axioms in $\mathcal{B}$ are assumed to be correct. In the context of ontology debugging, the remainder of axioms in $\mathcal{O}$ are considered potentially faulty [(Rodler 2015, p. 27)].*

Background knowledge, then, constitutes axioms that the oracle or knowledge engineer knows to be true before starting with testing. In the OntoDebug tool (fully introduced in Section 2.7), 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. This serves a two-fold purpose: firstly, it means that calls to the reasoner would return back *inconsistencies* if an unsatisfiable concept exists as the unsatisfiable concept would always have individuals associated with it. Secondly, it makes the reasonable and

Stellenbosch University https://scholar.sun.ac.za

generalisable assumption (that we would want to be captured in background knowledge to an ontology) that all concepts in the ontology should be able to be instantiated by individuals.

Positive and negative test cases then 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).

Axioms in positive or negative test cases can either be explicitly stated axioms that the knowledge engineer now through their understanding knows is (in)correct, or implicitly derived axioms that the knowledge engineer does (not) want to be entailed in future.

Definition 16. ***Positive test cases*** *(aggregated in the set P) correspond to desired entailments of the correct (repaired) ontology, $\mathcal{O}$ along with the background knowledge $\mathcal{B}$. Each test case $p \in P$ is a set of axioms over language $\mathcal{L}$. 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 $\mathcal{O}$ integrated with $\mathcal{B}$ [(Rodler 2015, p.27)].*

Definition 17. ***Negative test cases*** *(aggregated in the set N) represent undesired entailments of the correct (repaired) ontology $\mathcal{O}$, along with the background knowledge $\mathcal{B}$. Each test case $n \in N$ is a set of axioms over language $\mathcal{L}$. 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 $\mathcal{B}$ [(Rodler 2015, p.27)].*

To continue with our example – generally, the background knowledge would be the knowledge asserted in the Abox axioms. In the case of our ontology, $\mathcal{O}$, it would be the following statement:

$$\mathcal{A}_{\mathcal{O}} = \left\{ \mathsf{Staff(Maggie)} \right\}$$

A positive test case, something that the knowledge engineer would want to be implicitly entailed or explicitly asserted, could in this case be the following entailment from ontology $\mathcal{O}$:

Stellenbosch University https://scholar.sun.ac.za

$\mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.PublicInformation}$

If the knowledge engineer is confident that there are certain asserted axioms that must hold, they can also add the following explicitly asserted statements to the set of positive test cases:

$\mathsf{User} \sqsubseteq \exists \mathsf{accessTo.PublicInformation}$

$\mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.PrivateInformation}$

A negative test case, something that the knowledge engineer would *not* want to be implicitly entailed or explicitly asserted, could in this case be the following entailment from ontology $\mathcal{O}$:

$\mathsf{Staff} \sqsubseteq \neg \mathsf{User}$

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 18. *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)*** *[(Schekotihin et al. 2018, p.6)].*

Definition 19. *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 [(Schekotihin et al. 2018, p.6)].

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 (Rodler 2015, p.6-7).

In the example ontology, a subset of axioms that constitute the diagnosis can now be identified:

Stellenbosch University https://scholar.sun.ac.za

$$\mathcal{O}^d = \left\{ \begin{array}{l} \top \sqcap \exists \mathsf{accessTo.PublicInformation} \sqsubseteq \mathsf{PublicInfoConsumer} \\ \top \sqcap \exists \mathsf{accessTo.PrivateInformation} \sqsubseteq \mathsf{PrivateInfoConsumer} \\ \mathsf{PrivateInfoConsumer} \sqsubseteq \neg \mathsf{PublicInfoConsumer} \end{array} \right\}$$

From the now-limited output, it is easier for the knowledge engineer to identify the required repair – in this case, the deletion of $\mathsf{PrivateInfoConsumer} \sqsubseteq \neg \mathsf{PublicInfoConsumer}$. To prevent this axiom from being asserted in future, the knowledge engineer may then wish to add this axiom as a negative test case.

The benefit of setting up positive and negative test cases, and adding background knowledge, is twofold: Firstly, by setting up test cases test-driven ontology development is adhered to. By adhering to test-driven ontology development, test cases act as preventative controls to prevent bugs or previously discovered unwanted entailments or axioms from again creeping into the ontology in later development iterations (Schekotihin et al. 2018). Secondly, because wanted and unwanted entailments are explicitly stated, the scope of possibly faulty axioms is limited, thus making it easier to find the axiom(s) that require repair.

Generally, once test cases and background knowledge have been used to limit the number of possibly faulty axioms, debugging tools then aim to determine the axioms which are most probable to be faulty (Rodler 2015, p.7). The probability of a diagnosis to be faulty is generally determined by external criteria (e.g. fault probabilities of logical formulas in general or learning a user's usual fault patterns) – it is no longer the case that the fault is determined based on the content of the axioms in the ontology itself. Rodler argues however that this approach, although it further limits the scope of search for axioms to repair, could in fact introduce more faults in an ontology: selecting an incorrect diagnosis for repair could lead to unexpected entailments, desired entailments that fall away or future faults (2015, p. 7). Furthermore, he notes that in reality, ontologies with many axioms are likely to have many minimal diagnoses – and these could take a long time to sift through, set up test cases manually, and rerun the reasoner iteratively.

Rodler's (2015) 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.

Stellenbosch University https://scholar.sun.ac.za

Definition 20. *Let* $\boldsymbol{Ax}$ *be a set of axioms and* $ans : \boldsymbol{Ax} \rightarrow P, N$ *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. $ans(ax) = N \rightarrow \mathcal{O}^* \cup \mathcal{B} \not\models ax$

[(Schekotihin et al. 2018, p.7)].

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 21. *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* $\langle \mathcal{O}, \mathcal{B}, P \cup Q^P_{ans}, N \cup Q^N_{ans} \rangle$ *[(Schekotihin et al. 2018, p.8)].*

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.

Rodler (2015) developed algorithms for computing which query to pose to the user first, based on leading diagnoses, and his solution further provides the user with control to change the parameter inputs to the algorithm. It is not within the scope of this thesis to investigate this in more detail, as our extension focuses on designing a solution which provides suggestions for repair with minimal impact *after* the query-generation module has run – it is nonetheless important to briefly mention, as it shows rigour and flexibility in the base theory from which we are working, which would allow for fast, scalable ontology debugging.

2.7. Interactive ontology debugging with OntoDebug in Protégé

The above theory has already been implemented in a Protégé 5.0 plug-in by Rodler (2015) as the OntoDebug plug-in. The following sets out the purpose of the plug-in and will briefly

Stellenbosch University https://scholar.sun.ac.za

outline the flow and user interface of the plug-in. This is done as in Chapter 3, along with the foundational theory underlying our extension, we will provide an understanding of the desired flow which is inspired by the OntoDebug and DIP (see Section 2.4) plugins.

For this section, we will continue working with the example ontology that we were working with in Section 2.6. (For a primer on Protégé 5.0, please refer to Section 2.4):

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

$$\mathcal{A}_{\mathcal{O}} = \left\{ \mathsf{Staff(Maggie)} \right\}$$

Figure 9 shows the input ontology, which is the above set of axioms, together with some more classificatory axioms, implemented in a test ontology.

At this stage, certain axioms which the user knows to be background knowledge can be added. Figure 10 shows that in our example, we have added Abox axioms as background knowledge (if these are not explicitly added, OntoDebug will automatically generate instances associated with the concepts).

When the tool is run, a list of repairs (axioms which, when deleted, could effectuate a repair) is provided (refer to figure 11). At this stage, the list of possible repairs is quite extensive.

To limit the amount of possibly faulty axioms that the knowledge engineer needs to sift through, queries are systematically presented to the knowledge engineer where they need to confirm whether the axioms are true (+) or false (-) (refer to figure 12). The knowledge

Stellenbosch University https://scholar.sun.ac.za

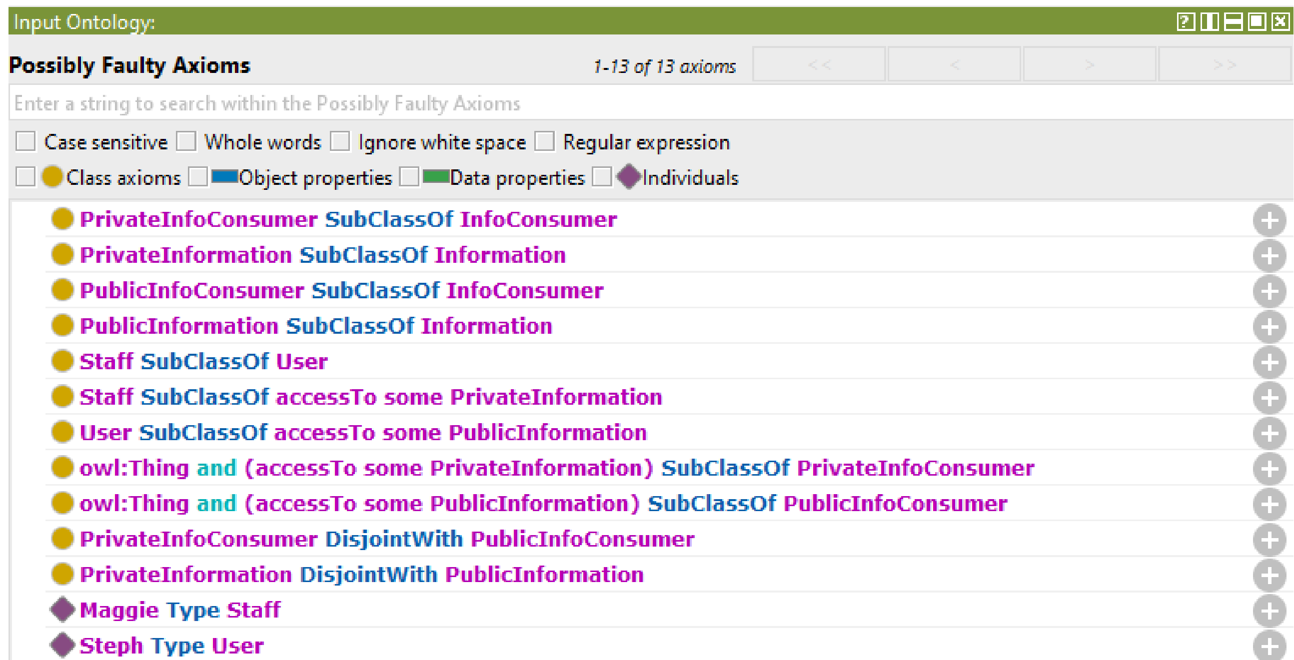


Figure 9 Input ontology used for illustration of OntoDebug.

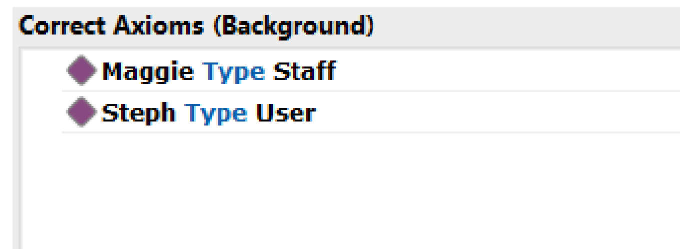


Figure 10 Setting background knowledge in OntoDebug.

engineer as the oracle has to answer which axioms they would like to be entailed (+) or not (-).

Once answers are submitted, these form part of the test cases within the session. When answers are submitted, the list of repairs will also grow smaller. Until a minimal number of repairs exist, further queries will be asked to the knowledge engineer.

Importantly, throughout this process, the reasoner runs to allow queries to be presented to the user iteratively all the while identifying incoherence in the knowledge base (refer to figure 14). It is this aspect of it that makes this tool's approach interactive in nature. It is by presenting queries to the user systematically, from which test cases are built, that allows

Stellenbosch University https://scholar.sun.ac.za

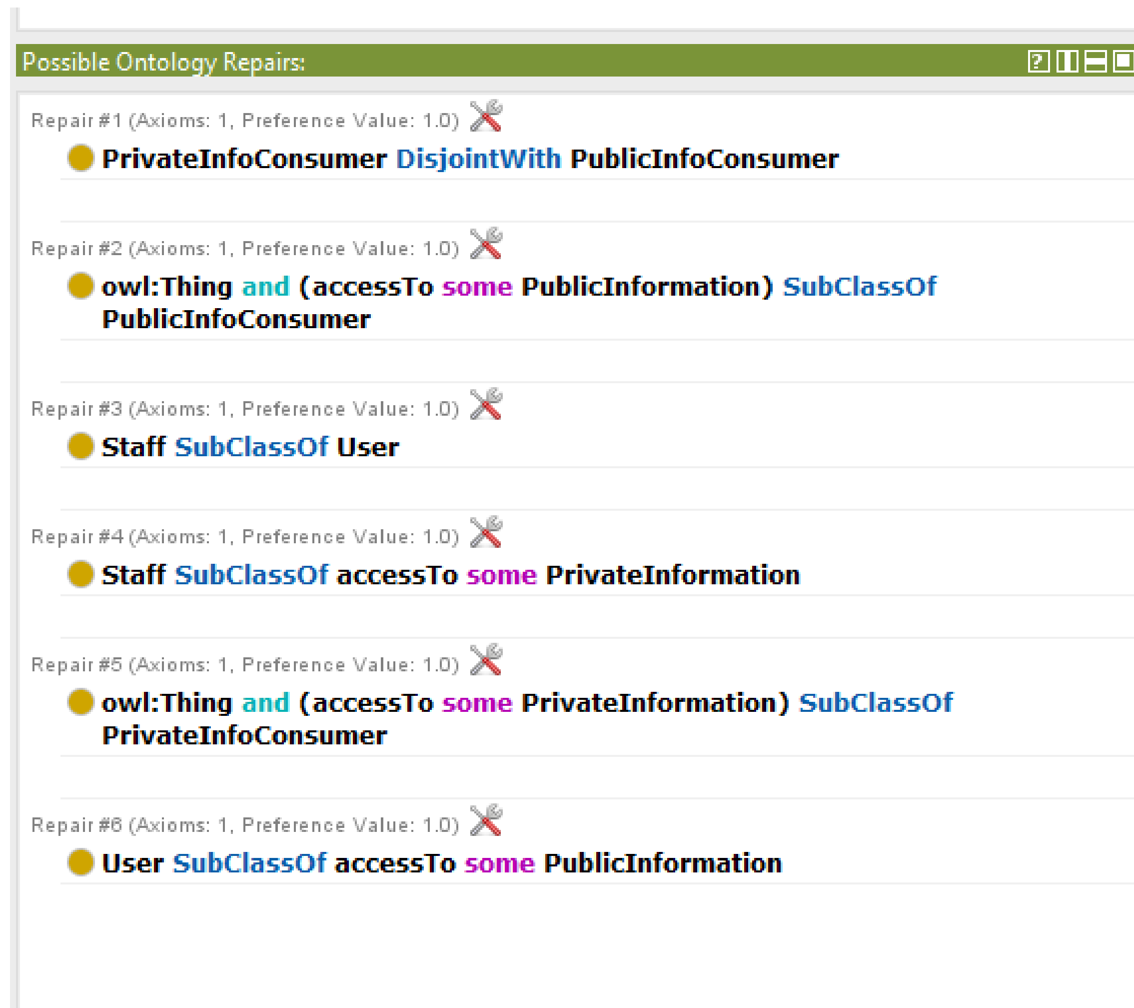


Figure 11 OntoDebug repair candidates – axioms which would, when deleted, effectuate a repair.

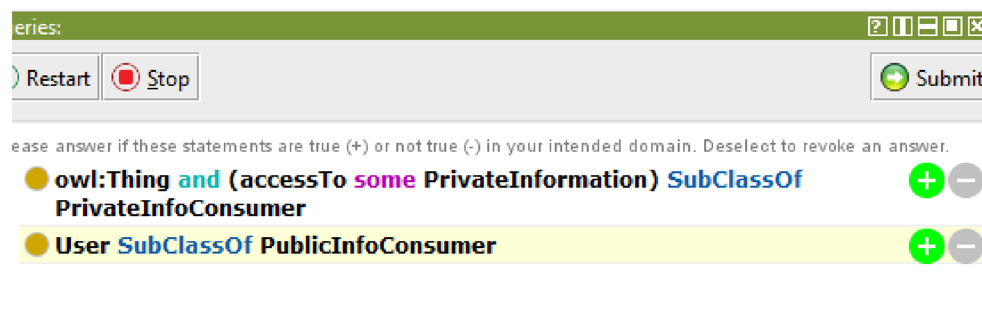


Figure 12 OntoDebug queries (first iteration).

for the list of faults, and therefore repairs, to grow smaller thus making the knowledge base repair process more manageable.

Stellenbosch University https://scholar.sun.ac.za

Figure 13 In OntoDebug submitted answers become test cases; for each query answered at least one repair candidate is eliminated; if more than one repair still exists, further queries are posed to the knowledge engineer.

Figure 14 Iterative nature of OntoDebug.

The process of presenting queries, adding these queries to the test cases, and then systematically re-calculating which queries to present to the user is iterated until the minimal

Stellenbosch University https://scholar.sun.ac.za

repair is found. For each query that is answered, one DPI is eliminated. As there can only be a finite number of DPIs, the procedure will always terminate.

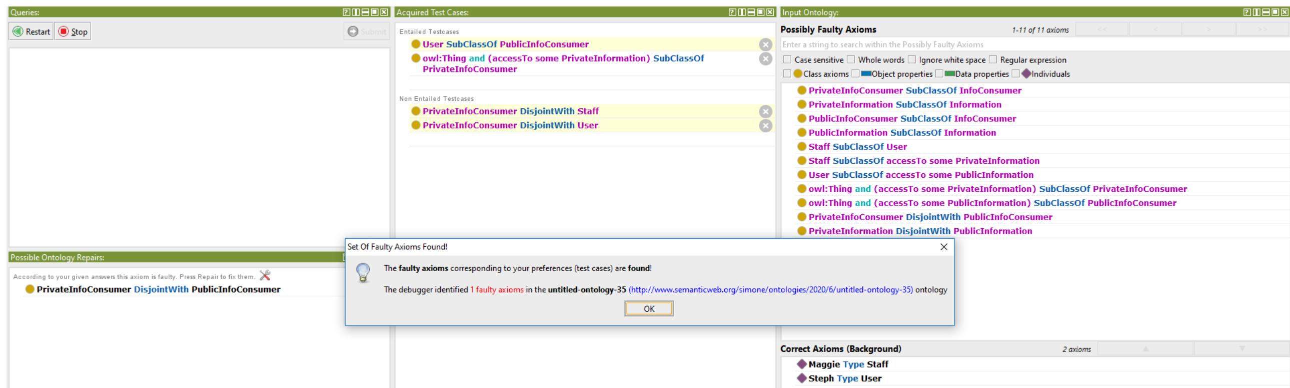


Figure 15 Following a process of iteration, a minimal repair candidate has been identified.

In this case, we see that it has correctly identified the disjointness condition between PrivateInfoConsumer and PublicInfoConsumer as the faulty axiom. It is then possible to select this axiom, and show explanations (justifications) which lead to this being the faulty axiom; from here the user can also edit or delete the axiom. To resolve this particular inconsistency, the disjointness axiom needs to be deleted.

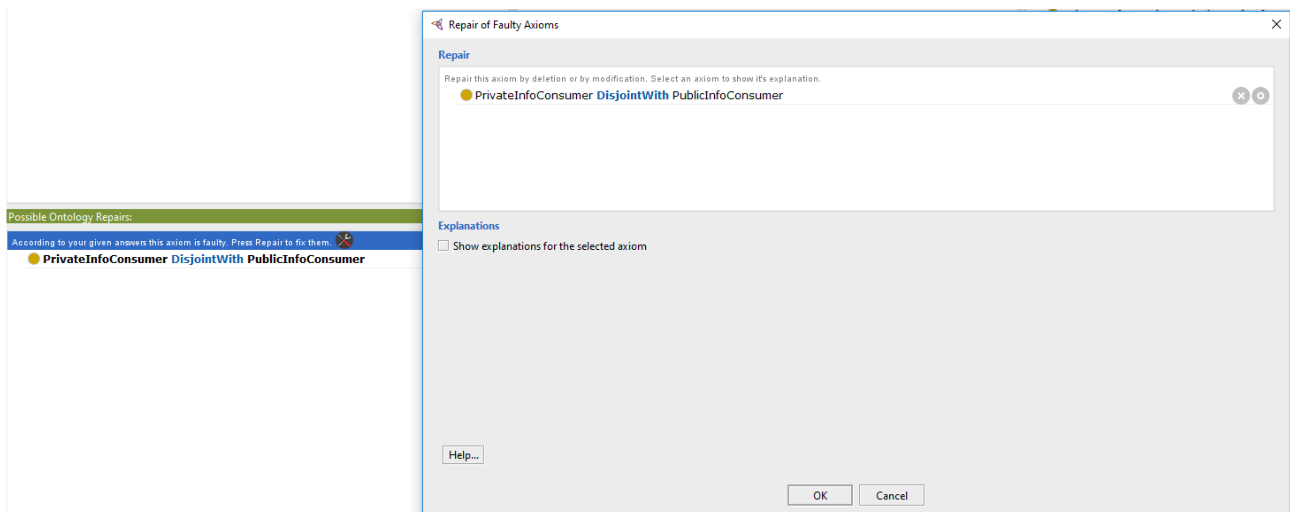


Figure 16 OntoDebug's repair screen.

Stellenbosch University https://scholar.sun.ac.za

At this stage, the reasoner is still running. Once the axiom is deleted or modified, again a call will be made to the reasoner to determine whether the ontology is now coherent and consistent. In this case, coherency and consistency has been reached.

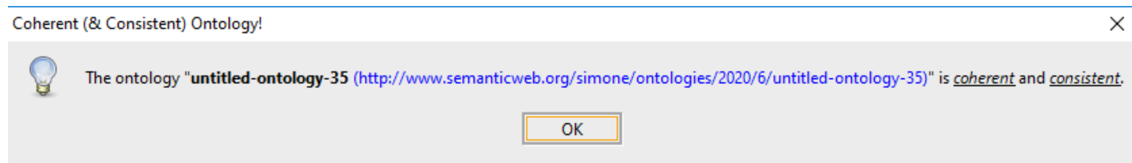


Figure 17 OntoDebug – repair success screen.

Through iteratively and systematically presenting the oracle/ knowledge engineer with queries that feed in to define positive and negative test cases, the scope of the search for the faulty axioms, along with the possible repairs, has been successfully narrowed down, and the knowledge engineer is able to fix the issue by deleting the disjointness axiom between two concepts.

Stellenbosch University https://scholar.sun.ac.za

Chapter 3

Incorporating Systematic Weakening into Interactive Ontology Debugging

In the first section of this chapter we will show that in some cases, particularly where we are dealing with faulty modelling due to axioms that are not necessarily incorrect, but that need to be weakened, OntoDebug returns unexpected results. Through example, we will show that this is likely to lead the knowledge engineer to delete valuable axioms – this results not only in a loss of valuable, nuanced knowledge, but can also cause further structural problems for the ontology as other axioms that depended on the axiom being deleted would also be affected.

In the following sections, we propose an extension to the the existing interactive ontology debugging methodology to enable the user to, at the click of a button, run further reasoning tasks that would systematically return suggestions for axiomatic weakening. In these sections we will thus provide the theoretical foundation for how the existing interactive ontology debugging methodology can be extended to allow the knowledge engineer to perform additional reasoning on the identified repair, and its related justifications, to obtain a recommendation on how the ontology can be repaired by weakening an existing axiom – this is done by lending from concepts in the Defeasible DL community discussed in the previous chapter. To follow this methodology, a minimal conflict set from the array of potential minimal conflict sets must be selected – we will provide the algorithm by which this can be done.

Through creating this extended methodology a new design pattern has been created, and can be used to suggest potential repairs to the knowledge engineer before the knowledge engineer starts with the interactive ontology debugging exercise – this will potentially reduce the time to find and rectify inconsistencies of the type where weakening of an axiom is required. Furthermore, we will assess how well the proposed extension to the OntoDebug tool works in practice with the existing interactive ontology debugging methodology – this will be done by assessing the way in which the methodology will flow if inconsistencies of different sorts are encountered alongside inconsistencies for which weakening is necessary.

Stellenbosch University https://scholar.sun.ac.za

Finally, we will also produce a user interaction roadmap, showcasing how the proposed methodology would be incorporated with the existing OntoDebug tool.

3.1. Unintuitive results obtained by interactive ontology debugging

For all its merits, using the interactive ontology debugging methodology, implemented in the OntoDebug tool, can at times lead to unintuitive results. This is due to the assumption that due to monotonicity of entailment in (classical) description logics, only deletion (and not expansion) of the KB can effectuate a repair of inconsistencies, incoherences and unwanted entailments (Rodler 2015, p. 30). In the original interactive ontology debugging approach, expansion can only ever be used when the knowledge engineer would like axioms in the positive test case to hold if they do not already hold. However, we have seen that there can often be (especially in the business, regulatory and legal domains) cases where rather than deleting an axiom to repair inconsistencies, we would want to modify the axiom and *weaken* it so that important, albeit nuanced or complicated, logic is maintained.

Let's revert back to the example we worked with in Section 2.3 to understand which unintuitive results are obtained when this is run through OntoDebug:

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

First, we create this ontology in Protégé 5.0:

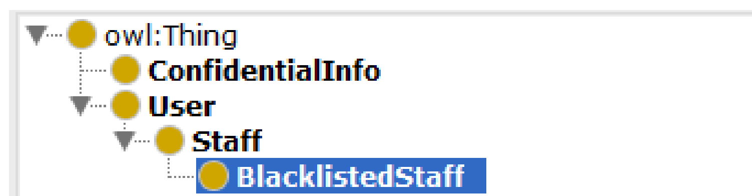


Figure 18 Ontology producing unintuitive result is created in Protégé 5.0.

Stellenbosch University https://scholar.sun.ac.za

Next, we start the interactive ontology debugging tool, OntoDebug. When it starts, we get a breakdown of all axioms which could possibly need repair (traditionally by deletion).

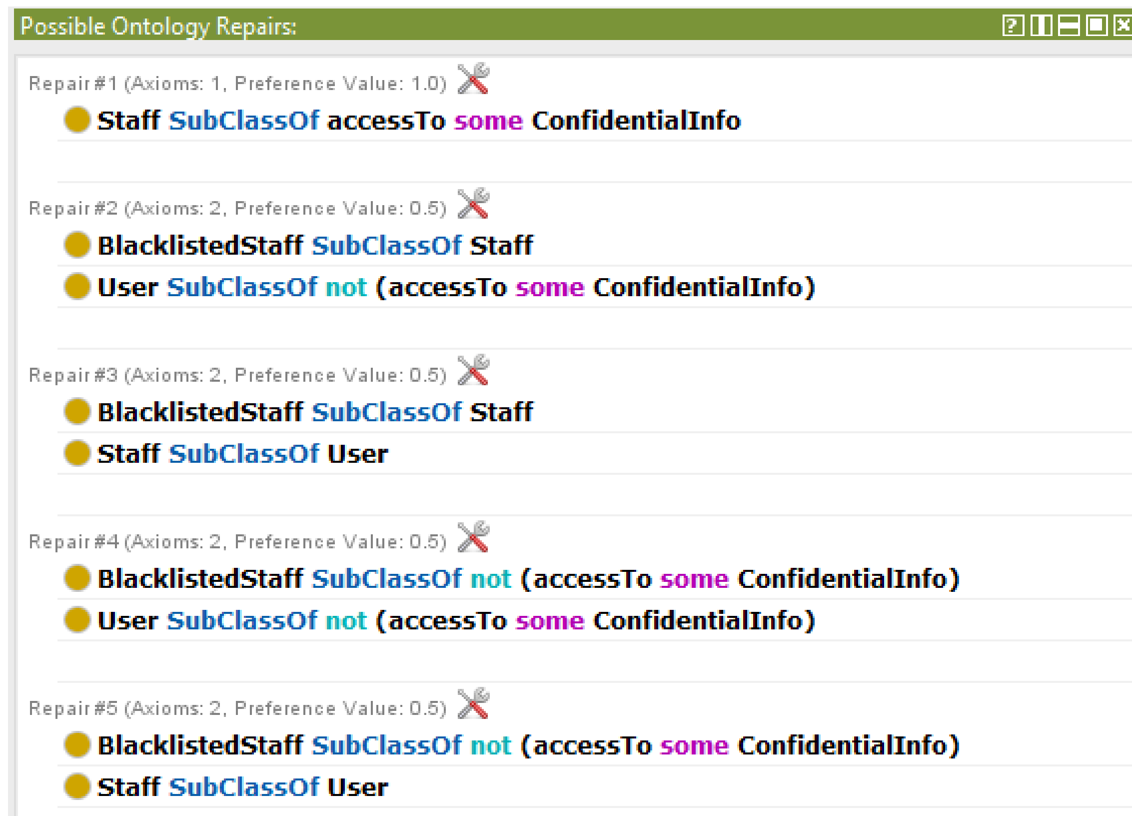


Figure 19 Iteration 1 of list of repair candidates is given for ontology producing unintuitive result.

As is customary of OntoDebug, the tool then presents the knowledge engineer with queries that need to be answered. The answers of these queries are fed through as test cases which trigger a new diagnosis to be calculated following the new DPI. In this case, we do want it to always be entailed that BlacklistedStaff do not have access to ConfidentialInfo and we do want it to be entailed that Staff is a particular type of user.

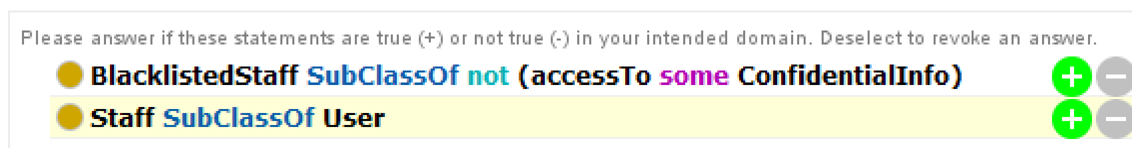


Figure 20 Iteration 1 queries generated of the ontology producing the unintuitive result.

Stellenbosch University https://scholar.sun.ac.za

When these queries are put through as positive test cases, the list of possible ontology repairs grows smaller due to the queries that have been answered.

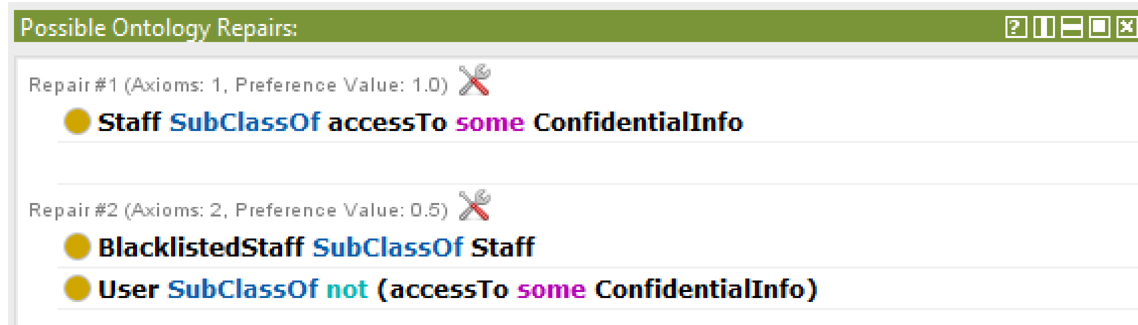


Figure 21 Iteration 2 possible repair candidates to the ontology producing the unintuitive result.

Again, a query is posed, as we have not yet reached a point where a minimal repair has been identified. To this query, we answer that we **do not** want it to be entailed that BlacklistedStaff is disjoint with Staff, as it is intuitive that a member of staff who is blacklisted is nonetheless a member of staff.

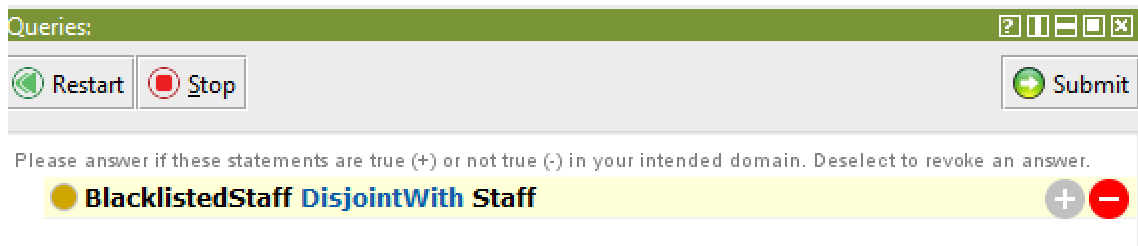


Figure 22 Iteration 2 query answering to the ontology producing unintuitive results.

Again, the possible repairs are calculated and this time OntoDebug completes its run as a minimal repair is found – the suggested repair is that a change should be made to Staff to say that they do not have access to ConfidentialInfo.

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 ConfidentialInfo,

Stellenbosch University https://scholar.sun.ac.za

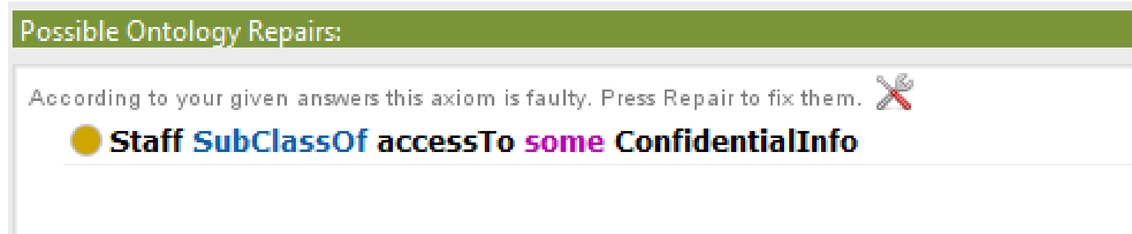


Figure 23 Unintuitive repair candidate is provided.

and we have also asserted that BlacklistedStaff (the concept subsumed by Staff) does not have access to 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. We know, therefore, that a more accurate repair would be to weaken axioms where the concept of User appears on the left hand side and to after that, weaken axioms where the concept of Staff appears on the left hand side. This ideal repair that we'd want to make to capture the nuances of user access would be as follows:

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

By weakening the concepts of User and Staff we are enabling the reading of the problematic axioms to be transformed to: 'a User *usually* does not have access to ConfidentialInfo' and 'Staff *usually* do have access to ConfidentialInfo' thus giving semantic structure to the underlying intuition causing us to believe this set of axioms to be coherent.

Ideally then, in the scenario where these sorts of bugs are picked up on, it should be possible to dig further for a result in OntoDebug should the initial result be unintuitive. It should be possible for the knowledge engineer to perform further debugging tasks on this specific axiom that is noted as a candidate for repair so that it can be determined how axioms related to this repair can instead be systematically weakened. The next section delves into the detail of how, from a design and high-level methodology perspective, this can be done.

Stellenbosch University https://scholar.sun.ac.za

3.2. Outline of methodology

We propose that Rodler's (2015) original interactive ontology debugging methodology be followed until an untintuitive result, as per Section 3.1, is obtained. This is done so that, should other types of inconsistencies – inconsistencies *not* requiring weakening as part of their repair – be identified, Rodler's original interactive ontology debugging methodology can be followed as is. This ensures that the existing functionality of the OntoDebug tool will remain unaffected by our proposed extension, but that we add capability to it that allows for more nuanced repair suggestions to be returned (for more details on how the existing OntoDebug functionality is affected by this extension, please refer to Section 3.5).

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, together, lead to this axiom being identified as a potentially faulty axiom. Convert all axioms to defeasible axioms.
2. **Determine a candidate axiom to weaken and a candidate concept with which to weaken it:** Use the ranking algorithm (refer to Section 2.3) to identify an axiom to weaken at level 0, and to identify a concept at level 1 with which to weaken the selected axiom.
3. **Weaken the relevant axiom:** Apply Cautious Monotonicity (CM) to weaken the selected axiom at level 0. This weakened result is what is then displayed to the knowledge engineer as a repair recommendation.
4. **Choose to accept or reject solution:** If the knowledge engineer accepts the repair recommendation which involves a more nuanced approach to repair (weakening axioms instead of deleting them), then the relevant axiom is weakened. If the knowledge engineer does not accept the repair recommendation, they can, as per the normal interactive ontology debugging route, exit and delete the faulty axiom.
5. **Repeat until done:** Whatever the knowledge engineer's choice, once they have made the choice, new faulty axioms will be calculated, and queries will be systematically presented to the user until again one faulty axioms are identified. This process is repeated until no faulty axioms are identified at which point the ontology is coherent

Stellenbosch University https://scholar.sun.ac.za

and consistent; or until the knowledge engineer exits the program at which point the reasoner is stopped and the ontology still remains incoherent/ inconsistent.

Let's apply the above methodology to the unintuitive repair result obtained in Section 3.1: in this section, the following ontology was considered:

$$\mathcal{O} = \begin{Bmatrix} \mathsf{User} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubseteq \mathsf{User} \\ \mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{BlackListedStaff} \sqsubseteq \mathsf{Staff} \\ \mathsf{BlackListedStaff} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo} \end{Bmatrix}$$

This ontology is incoherent (and, if individuals were added to the Staff or BlackListedStaff classes, inconsistent) because we are claiming that a User does not have access to ConfidentialInfo. Yet, we are also claiming that Staff, a type of User, has access to ConfidentialInfo. So, we are claiming that Staff both *have and do not have* access to ConfidentialInfo, which gives rise to the incoherence. To make matters more complicated, we are also claiming that Staff have access to ConfidentialInfo, but that BlackListedStaff (a type of Staff) do not have access to ConfidentialInfo. This means that BlackListedStaff both *do and do not* have access to ConfidentialInfo – this gives rise to an exception to the existing exception, Staff.

Following the interactive ontology debugging methodology as implemented in OntoDebug, the conclusion is reached that the following axiom is the axiom requiring repair (in this case deletion):

$\mathsf{Staff} \sqsubseteq \exists \mathsf{accessTo.ConfidentialInfo}$

However, as discussed in the previous section, we know that intuitively we do not want this axiom to be deleted. It is from this point onwards that the methodology described above is applied. Specifically applied to this case:

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. Convert all axioms to defeasible axioms so that the ranking algorithm can be applied in the next step.

Stellenbosch University https://scholar.sun.ac.za

*Note: It is possible that many different minimal conflict sets exist for one identified conflict. For more information on how the minimal conflict sets leading to the inconsistency should be chosen for processing, please refer to Section 3.3.

The following ontology is created:

$$\mathcal{O}' = \begin{Bmatrix} \mathsf{User} \sqsubsetsim \neg\exists \mathsf{accessTo.ConfidentialInfo} \\ \mathsf{Staff} \sqsubsetsim \mathsf{User} \\ \mathsf{Staff} \sqsubsetsim \exists \mathsf{accessTo.ConfidentialInfo} \end{Bmatrix}$$

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 (explained in Section 2.3) is used on the above ontology $\mathcal{O}'$: central to the ranking formula is the notion of exceptionality (see Definition 10 in Section 2.3).
   (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.

   In accordance with Definition 10, the concept User is not exceptional w.r.t. $\mathcal{O}'$, and so the following statement is also not exceptional.

$$\mathcal{O}'^{0} = \begin{Bmatrix} \mathsf{User} \sqsubsetsim \neg\exists \mathsf{accessTo.ConfidentialInfo} \end{Bmatrix}$$

   This axiom is removed from $\mathcal{O}'$ with the remainder renamed $\mathcal{O}''$, and the ranking of the axiom is saved.

   (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.

   The new knowledge base would thus contain the following statements:

$$\mathcal{O}'' = \begin{Bmatrix} \mathsf{Staff} \sqsubsetsim \mathsf{User} \\ \mathsf{Staff} \sqsubsetsim \exists \mathsf{accessTo.ConfidentialInfo} \end{Bmatrix}$$

   Staff is now deemed to be unexceptional w.r.t. $\mathcal{O}''$ (see Definition 10), and thus this concept gets a ranking of 1. Consequently, both the above axioms get a ranking of 1.

For the temporary ontology $\mathcal{O}'$, there are no further statements to assess. Per the ranking algorithm, concepts now have the following ranking:

Stellenbosch University https://scholar.sun.ac.za

World order/ rank	Concept
0	User
1	Staff

Table 2 **First iteration concept ranking output.**

And axioms have the following ranking:

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

Table 3 **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: in the next iteration of this solution, we will see that whereas in this iteration User has a rank of 0, and Staff has a rank of 1, in the next iteration Staff will have a rank of 0 and BlackListedStaff will have a rank of 1. This is because at the stage when we are dealing with the second level of exceptionality between Staff and BlackListedStaff the conflict between User and Staff has already been resolved through weakening (explained in next step).

3. **Weaken the relevant axiom:** Next, the postulate of Cautious Monotonicity is applied to weaken the axiom at level 0. As referenced in Definition 5:

$$\text{(CM)}\ \frac{C \sqsubseteq\!\!\!\!\sim D,\ C \sqsubseteq\!\!\!\!\sim E}{C \sqcap E \sqsubseteq\!\!\!\!\sim D}$$

This reads: *if C is defeasibly subsumed by D and C is defeasibly subsumed by E then C and E are defeasibly subsumed by D*. We know that the weakened result we would like to get to has a format similar to that of the axiom below the line: $C \sqcap E \sqsubseteq\!\!\!\!\sim D$. In our case, the weakened result would be $\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubseteq\!\!\!\!\sim \neg\exists\mathsf{accessTo.ConfidentialInfo}$. Thus we find that in the postulate of Cautious Monotonicity, C can represent User,

Stellenbosch University https://scholar.sun.ac.za

D can represent $\neg\exists\mathsf{accessTo.ConfidentialInfo}$ and E can represent $\neg\mathsf{Staff}$:

$$(\text{CM})\ \frac{\mathsf{User} \mathrel{\sqsubseteq\!\!\!\sim} \neg\exists\mathsf{accessTo.ConfidentialInfo},\ \mathsf{User} \mathrel{\sqsubseteq\!\!\!\sim} \neg\mathsf{Staff}}{\mathsf{User} \sqcap \neg\mathsf{Staff} \mathrel{\sqsubseteq\!\!\!\sim} \neg\exists\mathsf{accessTo.ConfidentialInfo}}$$

The rule that is extrapolated here is thus that when using Cautious Monotonicity to apply weakening to the 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 the concept at level 1; the resultant conclusion (bottom axiom) is then the axiom showing the weakened result.

For the above rule to hold, the following proof is necessary to show that the concept at level 0 (User) is defeasibly subsumed by the negation of the concept at level 1 (Staff):

Lemma 2. *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 $C \mathrel{\sqsubseteq\!\!\!\sim} \neg E$ 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 11 that $C \sqcap \neg E$ is in the rational closure of $\mathcal{O}$. □

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 ranks of more 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.

4. **Choose to accept or reject solution:** The classical counterpart of the defeasible axiom obtained by applying Cautious Monotonicity ($\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubseteq \neg\exists\mathsf{accessTo.ConfidentialInfo}$) is what is then displayed to the knowledge engineer as a repair recommendation. This axiom is a weakening of the original classical axiom $\mathsf{User} \sqsubseteq \neg\exists\mathsf{accessTo.ConfidentialInfo}$.

Stellenbosch University https://scholar.sun.ac.za

Let us assume the knowledge engineer accepts the above solution, and the original ontology is updated to read as follows:

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

5. **Repeat until done:** This process is repeated.

We find that in the main ontology there is still an inconsistency present – that we are asserting that BlackListedStaff, being a specialised case of Staff both *do and do not* have access to ConfidentialInfo.

1. **Isolate the issue:** After each iteration, the sub-ontology that was created is deleted. Therefore, we can again 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.

   The following ontology is created:

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

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 (explained in Section 2.3) is used on the above ontology $\mathcal{O}'$: central to the ranking formula is the notion of exceptionality (see Definition 10 in Section 2.3).

   (a) Ranking of 0 – least exceptional: First we identify the concepts with a rank of 0. In this case this would be Staff. Then, the statements with a rank of 0 are identified.

   In accordance with Definition 10, the concept Staff is not exceptional w.r.t. $\mathcal{O}'$, and so the following statement is also not exceptional.

$$\mathcal{O}'^{0} = \left\{ \mathsf{Staff} \precsim \exists\mathsf{accessTo.ConfidentialInfo} \right\}$$

   This axiom is removed from $\mathcal{O}'$ with the remainder renamed $\mathcal{O}''$, and the ranking of the axiom is saved.

Stellenbosch University https://scholar.sun.ac.za

(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.

The new knowledge base would thus contain the following statements:

$$\mathcal{O}'' = \left\{ \begin{array}{l} \mathsf{BlackListedStaff} \sqsubsetsim \mathsf{Staff} \\ \mathsf{BlackListedStaff} \sqsubsetsim \neg\exists \mathsf{accessTo.ConfidentialInfo} \end{array} \right\}$$

$\mathsf{BlackListedStaff}$ is now deemed to be unexceptional w.r.t. $\mathcal{O}''$ (see Definition 10), and thus this concept gets a ranking of 1. Consequently, both the above axioms get a ranking of 1.

For the temporary ontology $\mathcal{O}'$, there are no further statements to assess. Per the ranking algorithm, concepts now have the following ranking:

World order/ rank	Concept
0	Staff
1	BlackListedStaff

Table 4 Second iteration concept ranking output.

And axioms have the following ranking:

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

Table 5 Second iteration axiom ranking output.

3. **Weaken the relevant axiom:** Next, the postulate of Cautious Monotonicity is applied to weaken the axiom at level 0. As referenced in Definition 5:

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

Stellenbosch University https://scholar.sun.ac.za

We apply the rule stipulated in the previous iteration: when using Cautious Monotonicity to apply weakening to the axiom at level 0, use the axiom as is for the first premise in the postulate; for the second premise, use the subsumed concept at level 0 subsumed by the negation of the concept at level 1; the resultant conclusion is then the axiom showing the weakened result.

By applying CM to our example specifically, we find that C represents Staff, D represents $\exists\mathsf{accessTo.ConfidentialInfo}$ and E represents $\neg\mathsf{BlackListedStaff}$:

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

4. **Choose to accept or reject solution:** Let us assume the knowledge engineer accepts the classical counterpart of the above solution, and the ontology is updated to read as follows:

$$\mathcal{O}' = \left\{ \begin{array}{l} \mathsf{Staff} \sqcap \neg\mathsf{BlackListedStaff} \sqsubseteq \exists\mathsf{accessTo.ConfidentialInfo} \\ \mathsf{BlackListedStaff} \sqsubseteq \mathsf{Staff} \\ \mathsf{BlackListedStaff} \sqsubseteq \neg\exists\mathsf{accessTo.ConfidentialInfo} \end{array} \right\}$$

5. **Repeat until done:** Again it is checked whether all inconsistencies are resolved, and as they are, the debugging process stops running.

3.3. Selecting minimal conflict sets and choosing repair axioms

An important step during the above process is the selection of an axiom to repair, along with the selection of a minimal conflict set on which to apply the ranking algorithm on, and generate the weakened axiom from. First we explore how suggestions for selecting the correct minimal conflict set can be provided to the knowledge engineer once a repair has been identified. Up until this point, we have worked with an example where the suggested repair contains only one axiom – a repair can however contain more than one axiom, and therefore we explore how repairs themselves can be presented to the knowledge engineer so that axioms leading to root cause inconsistencies are resolved first.

Stellenbosch University https://scholar.sun.ac.za

3.3.1. Highlighting the most relevant minimal conflict sets: Let's continue to work with the example provided in the previous section:

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

As explained in Section 3.1, when following the standard OntoDebug methodology for the above example, the following axiom is provided as the axiom that requires repair:

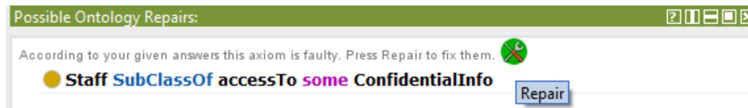


Figure 24 Unintuitive axiom to repair.

When the knowledge engineer clicks on the 'Repair' button, the option is provided to show the justifications leading to the result that certain classes are a subclass of Nothing (i.e. incoherent; in the debugging context, these are also known as minimal conflict sets).

With reference to figure 25, three minimal conflict sets are returned. At this stage of the debugging process, our extension does not automatically pick a justification to work from – this needs to remain up to the knowledge engineer as it is not necessarily the case that all inconsistencies should always be solved through weakening.

We do however propose that, it would be useful to highlight the minimal conflict set to the knowledge engineer which would be best to apply the ranking and weakening on. In this case, the first minimal conflict set is the minimal conflict set that would be most apt to apply weakening to, for the following reasons:

1. Comparing the first and the second minimal conflict sets, we find that the first minimal conflict set would allow for concepts at the root of the issue to be weakened first – i.e. applying the ranking algorithm and then weakening the axiom at level 0 (refer to

Stellenbosch University https://scholar.sun.ac.za

Repair of Faulty Axioms

Repair

Repair this axiom by deletion or by modification. Select an axiom to show it's explanation.

Staff SubClassOf accessTo some ConfidentialInfo

Explanations

Show explanations for the selected axiom

The selected axiom is responsible for an inconsistency

Show regular justifications | All justifications

Show laconic justifications | Limit justifications to 2

Explanation for: owl:Thing SubClassOf owl:Nothing

1) Staff SubClassOf accessTo some ConfidentialInfo — In ALL other justifications
2) _:genid2 Type Staff — In NO other justifications
3) User SubClassOf not (accessTo some ConfidentialInfo) — In 1 other justifications
4) Staff SubClassOf User — In 1 other justifications

1) _:genid1 Type BlacklistedStaff — In 1 other justifications
2) BlacklistedStaff SubClassOf Staff — In 1 other justifications
3) Staff SubClassOf accessTo some ConfidentialInfo — In ALL other justifications
4) BlacklistedStaff SubClassOf not (accessTo some ConfidentialInfo) — In NO other justifications

Explanation for: owl:Thing SubClassOf owl:Nothing

1) _:genid1 Type BlacklistedStaff — In 1 other justifications
2) BlacklistedStaff SubClassOf Staff — In 1 other justifications
3) Staff SubClassOf accessTo some ConfidentialInfo — In ALL other justifications
4) User SubClassOf not (accessTo some ConfidentialInfo) — In 1 other justifications
5) Staff SubClassOf User — In 1 other justifications

Figure 25 List of justifications associated with unintuitive repair candidate.

Section 3.2), the axiom with left hand User is weakened, rather than the axiom with left hand Staff.

This aligns with Rodler's (2015) overall aim of first fixing those axioms which lie at the root of the problem first. His motivation for this is that more specific concepts might be unsatisfiable due to more general concepts which they are subsumed by being unsatisfiable. Thus, by solving the root cause inconsistencies first, it may be that the more specific concepts that were reliant on the more general concepts may then become satisfiable simultaneously.

2. Comparing the first and the third minimal conflict sets, we find that the first minimal conflict set contains only Staff and User on the left hand side, whereas the third minimal conflict set also contains BlacklistedStaff on the left hand side. Because our algorithm would rank it as rank 2, this concept would not be of use to the rest of our algorithm, as the algorithm only uses concepts on level 0 and level 1.

Stellenbosch University https://scholar.sun.ac.za

For both the second and the third conflict sets above, the proposed methodology explained in Section 3.2 would still ultimately yield the correct results. However from a user experience perspective, the results may be confusing to the knowledge engineer, as it would mean that axioms are not weakened in a systematic fashion: more exceptional axioms (like those containing Staff or BlacklistedStaff) would be weakened first, rather than axioms associated with the root cause first. I.e. rather than going in an ordered fashion through the least exceptional axioms (those containing User on the left hand side), to the more exceptional axioms (those containing Staff and BlacklistedStaff on the left hand side), the knowledge engineer may trail through the path of exceptionality in a haphazard manner.

Therefore, the minimal conflict sets most suited to weakening should be highlighted to the knowledge engineer, and should be displayed first. To achieve this aim, the following two rules can be applied:

1. Once a user selects a specific axiom to investigate further, the minimal conflict set(s) that have an individual or generated individual belonging to a more general type (in this case, Staff) should be displayed first, and highlighted as suitable for weakening. If OntoDebug finds a concept to be unsatisfiable, it will automatically create a dummy individual associated with the concept (if an individual does not already exist) so that an ontology inconsistency is flagged.
2. If more than one minimal conflict set is present for a given individual or generated individual, then the conflict set with fewer concepts on the left hand side should be displayed first. Research has shown that smaller justification sets are easier for knowledge engineers to digest and interpret (Horridge et al. (2011, 2013), Kalyanpur et al. (2006)), thus facilitating greater understanding and engagement throughout the debugging lifecycle. Furthermore, having fewer axioms ensures that the ranking algorithm operates more efficiently (as fewer concepts need to be ranked).

If the knowledge engineer adheres to the prompts indicating which minimal conflict set is most apt to solve first, the benefits are that (a) root cause inconsistencies will be solved first, and (b) fewer concepts will be fed to the ranking algorithm, which may show some performance-related improvements in addition to being easier to interpret.

3.3.2. Choosing repair axioms: Up until now, the example that we have worked with will provide only one axiom as an axiom to repair. When adding the below additional

Stellenbosch University https://scholar.sun.ac.za

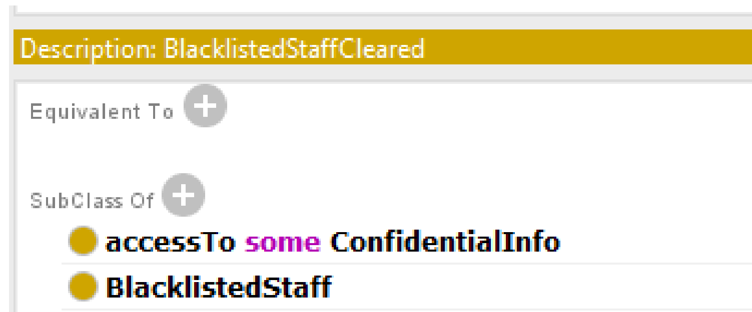


Figure 26 OntoDebug behaviour when adding additional level of exceptionality.

axiom to our working example, OntoDebug returns two axioms as the axioms requiring reparation:

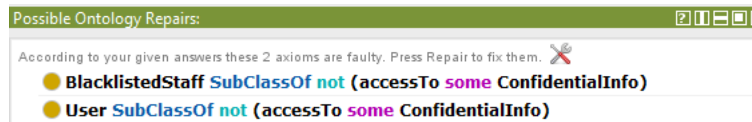


Figure 27 OntoDebug behaviour when adding additional level of exceptionality: two repair candidates are listed.

This happens for the following reason: OntoDebug, being originally created as a debugging environment catering for monotonic exceptions only, does not rank concepts by nature. Therefore, when we ran OntoDebug on our initial example (with only User, Staff and BlacklistedStaff), it found the axiom with Staff to be the 'odd one out', so to speak, as it is the only concept in the concept hierarchy chain that allowed for individuals to have accessTo.ConfidentialInfo. The other two concepts, User and BlacklistedStaff, *cohere* more with each other as they are in the same concept hierarchy chain, and do not allow for any of their individuals to have accessTo.ConfidentialInfo.

However, when the above axiom defining BlacklistedStaffCleared is added to the concept hierarchy chain, and now coheres with the concept of Staff, Staff is now no longer the 'odd one out', and so OntoDebug searches for the root cause inconsistencies – now being the axioms in figure 27.

This means that for every second level of exceptionality that is added, the list of repair axioms relating to the chain of exceptionality will grow by one further repair axiom. Therefore, it is necessary to order the repair axioms so that the repair axioms at the root

Stellenbosch University https://scholar.sun.ac.za

cause of the problem are solved first. The motivation for this follows the same pattern of thought that Rodler (Rodler 2015) adhered to when creating his querying algorithm, and provides more order when the user is traversing through multi-level exceptions.

The answer we propose to this is simple: for each repair, sort the axioms based on the generality of the left hand side concept – the more general the concept on the left hand side, the higher on the list the axiom will be.

3.4. Use of systematic weakening as a design pattern

Rodler's (2015) OntoDebug supports many different debugging activities: arguably, the main functionality of it lies in guiding the user throughout the debugging process by posing queries to the user which, when answered, narrows down the list of repairs. Even before the interactive ontology debugging methodology is followed, however, the tool also presents the user with the list of potential candidates for repair. By listing out the repairs at the very beginning of the ontology debugging process – before the full interactive ontology debugging methodology is followed – users can find and correct faults sooner than if they were to go through the interactive ontology debugging methodology.

In this section, we show that where multi-level exceptions are concerned, either the model-based diagnosis approach or a more heuristic approach can be followed to pre-generate potential repairs at the outset. The repairs can be viewed as a design pattern that is employed to resolve inconsistencies in both cases.

3.4.1. Using axiomatic weakening in a heuristic approach to debugging: Thus far, we have worked with model-based diagnosis (refer to Section 2.5) for debugging the ontology. That is, we have identified that the ontology is in an undesirable state (it is inconsistent/ incoherent), and we are aiming to get to the root of the issue to solve the inconsistency/ incoherence so that the ontology is again in the desired state (coherent and consistent). The heuristic approach to debugging tries to find common patterns of faulty ontology modelling and presents suggestions for repairs based on this (Rodler et al. 2019). 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

Stellenbosch University https://scholar.sun.ac.za

each inconsistency before returning a result. The drawback of using only the heuristic approach is that the repair suggestions may be incomplete, as it will only flag inconsistencies following a certain pattern (Rodler 2015).

The idea of using ontology design patterns goes hand-in-hand with the heuristic approach to debugging: Gangemi and Presutti (2009) 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 22. *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: $G \sqcap \neg H \sqsubseteq I$.

Gangemi and Presutti (2009) list different categories of design patterns (*logical, architectural, transformation, reasoning, correspondence, refactoring, mapping, presentation*). From their listing, the above can be described as a *logical* ontology design pattern (OP) as logical OPs solve problems of expressivity. They help to solve design problems where "the primitives of the representation language do not directly support certain logical constructs" (2009, 5). In this case, the primitives of classical DLs do not support the notion of a concept being *usually* subsumed by another concept – classical DLs do not allow for defeasible subsumption, $\precsim$.

Stellenbosch University https://scholar.sun.ac.za

Gangemi and Presutti (2009) argue that it is vital to maintain a modular catalogue and repository of design patterns, and they show that this repository of design patterns can be incorporated into the ontology design environment to enable more standard design, thus enabling reusability and readability of an ontology. We posit that this ontology design pattern repository can be equally valuable in the debugging environment, especially as design patterns often stem from modelling problems. In such a repository, the above design pattern would have the following properties:

- **Name:** Axiomatic weakening
- **Intent:** To weaken the axiom with the most general concept (with the lowest rank) on the left hand side by left-strengthening the concept on the left hand side by adding a conjunction with the negation of the exceptional concept, as follows: $G \sqcap \neg H \sqsubseteq I$. This enables concepts that are intuitively subsumed by other concepts to remain in that subsumption relation, while also allowing for subsumed concepts to have a more specific relation to another concept which is exceptional had the weakening not taken place.
- **Diagram – recurrent ontology design problem:**

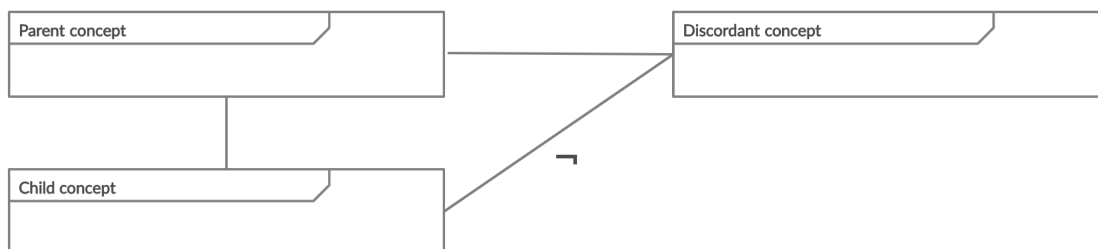


Figure 28 Recurrent ontology design problem.

- **Diagram – ontology design pattern/solution:**

Stellenbosch University https://scholar.sun.ac.za

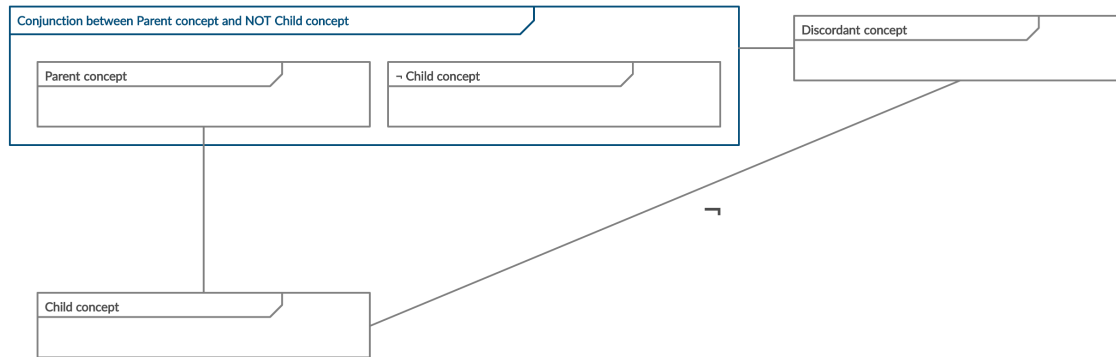


Figure 29 Ontology design pattern/ solution.

- **Elements:**
  - —`Parent concept` – anything that subsumes another object. May be a complex concept.
  - —`Child concept` – anything that is subsumed by another object. May be a complex concept.
  - —`Discordant concept` – The concept that the `Parent concept` is subsumed by, but the `Child concept` is not subsumed by, thus leading to a direct contradiction in the `Child concept`. May be a complex concept.

A heuristic approach to debugging is beneficial in the case where the knowledge engineer is dealing with a large ontology and where debugging needs to be performed swiftly, as minimal conflict sets do not need to be computed for each inconsistency.

The axiomatic weakening design pattern can only deal with simple cases of axiomatic weakening – more complex cases will be examined in the next chapter where these approaches are evaluated. Furthermore, as mentioned previously, the heuristic approach can return incomplete results due to only looking for a specific faulty modelling pattern.

3.4.2. Using axiomatic weakening in a model-based diagnosis approach to debugging: Whereas a heuristic approach, as explained above, can be used to quickly highlight how inconsistencies can be resolved, whilst retaining as much knowledge as possible, the same design pattern can also be employed when using the model-based diagnosis approach.

Figure 30 shows an example of the different axioms listed for repair, before the knowledge engineer answers all relevant queries to eventually get to a single repair/ diagnosis.

Stellenbosch University https://scholar.sun.ac.za

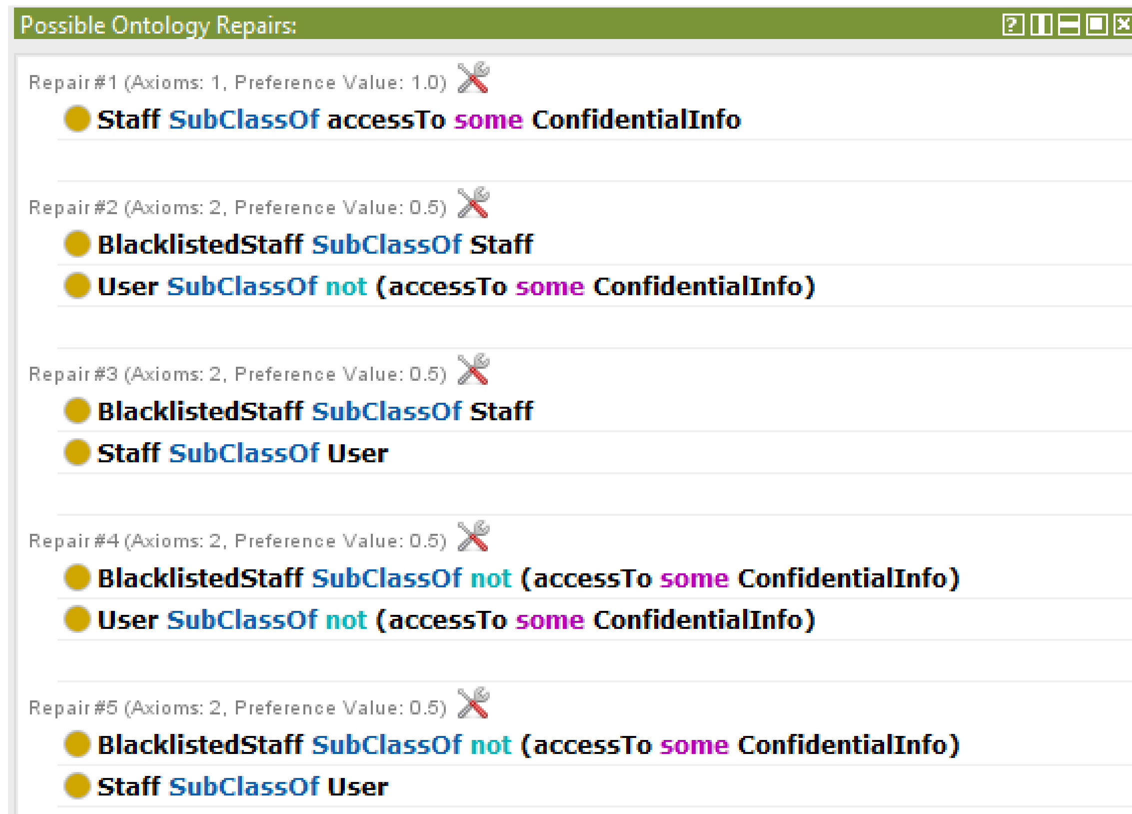


Figure 30 Iteration 1 of list of repairs is given for ontology producing unintuitive result.

Currently, the output only shows which candidate axioms are in need of repair, but it does not show *how* the axiom could be repaired.

Each of these repairs have a set of minimal conflict sets associated with them. We propose that for each repair candidate, the same logic can be followed as in Section 3.3 to identify a minimal conflict set suitable for weakening. Then, the algorithm explained in Section 3.2 can be used to obtain the weakened axiom which is then displayed to the user.

The benefit of following the model-based diagnosis approach to debugging is that it will return more complete results, as it evaluates the entire ontology for inconsistencies/ incoherences. Because the entire ontology is evaluated, and because minimal conflict sets need to be computed for each inconsistency/ incoherence, the approach is however less efficient and especially for larger ontologies can take more time to execute.

We recommend using the heuristic approach alongside a model-based diagnosis approach: this has the benefit that the user can view repair suggestions quickly via the heuristic

Stellenbosch University https://scholar.sun.ac.za

approach's output. For more extensive debugging, the model-based diagnosis approach can then be used to compute repair suggestions for each axiom listed as a repair candidate.

3.5. Integration with OntoDebug

From our above recommendations, there are two implications that our proposals have on the OntoDebug tool: firstly, the interactive ontology debugging workflow will need to be extended so that a knowledge engineer can run further reasoning tasks to understand how a specific repair candidate can be weakened; secondly, before the interactive ontology debugging workflow is followed by the knowledge engineer, the initial reasoning tasks performed to show the repair candidates will need to be amended to incorporate the proposed design pattern.

3.5.1. Integration with existing interactive ontology debugging workflow: Our recommendation fits in as an *extension*: the bulk of the initial workflow stays the same; it is only at the point where the knowledge engineer would like to investigate why a specific axiom is listed as a repair candidate, and how the axiom can be weakened, that our extension kicks in.

Figure 31 shows the worklfow that is followed. The boxes and lines in blue represent the original OntoDebug steps, and the boxes and lines in green represent our extension.

The original workflow elements are as follows:

1. The knowledge engineer starts running OntoDebug – HermiT is used as the reasoner, as it continues running even when inconsistencies are detected, thus enabling interactive debugging to take place.
2. The reasoner checks whether there are any inconsistencies or incoherences – for concepts that are unsatisfiable and have no instances associated with them, OntoDebug will generate an instance for that concept.
3. If no inconsistencies or incoherences are detected, the workflow ends. If inconsistencies or incoherences are detected, the workflow calculates the diagnoses (also referred to as repair candidates) using the faulty ontology, positive test cases, negative test cases and background knowledge (refer to Section 2.6 for more information on these concepts).

Stellenbosch University https://scholar.sun.ac.za

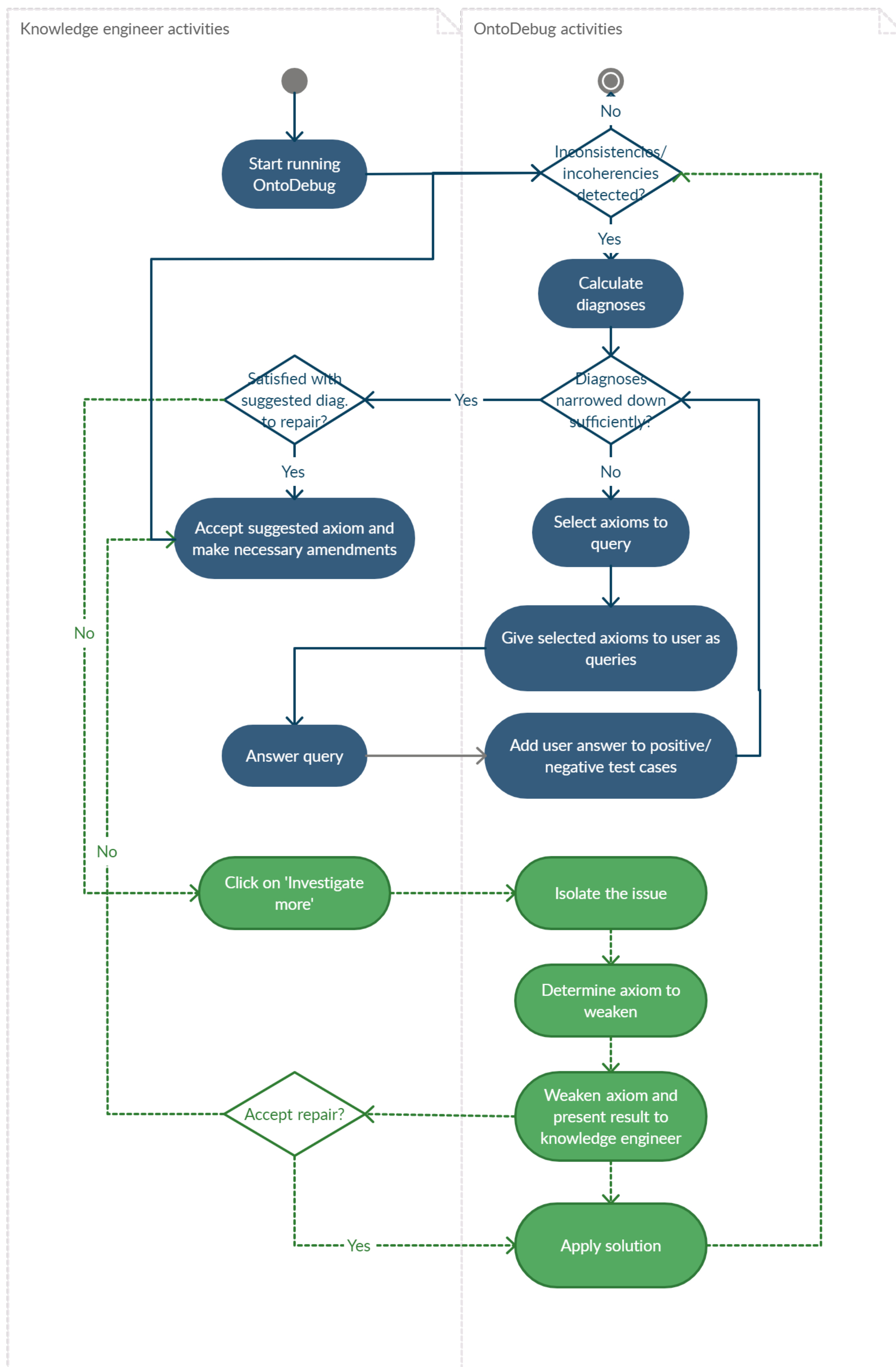


Figure 31 Activity diagram: extension to OntoDebug – green boxes and lines represent proposed OntoDebug extension.

Stellenbosch University https://scholar.sun.ac.za

4. Once the diagnoses are calculated, the knowledge engineer can at any stage investigate the diagnoses to understand why inconsistencies are occurring, and to fix them. The next subsection explains how our methodology integrates with the initial repair-generation tasks.
5. Diagnoses are gradually trimmed down as OntoDebug will pose a query to the user to build up test cases and eliminate diagnoses. Queries are explicit axioms already captured in the ontology that are systematically proposed to users, based on their being linked to a leading diagnosis, and the user needs to confirm whether the axiom holds (in which case it automatically becomes a positive test case) or not (in which case it automatically becomes a negative test case). Refer to Chapter 2.6 for more information on these concepts. With each query that is posed, at least one diagnosis will be eliminated thus making the number of diagnoses more digestible for the knowledge engineer. Queries are continually asked and answered until the results have been narrowed down sufficiently.
6. Once the diagnoses have been trimmed down sufficiently, the knowledge engineer can view them, along with their minimal conflict sets, and can choose to repair that specific axiom

   At this stage, the original OntoDebug workflow does not suggest how an axiom can be repaired by weakening. This is where the original OntoDebug worklfow stops – after the knowledge engineer has deleted or modified the repair candidate, again OntoDebug checks for inconsistencies. If none are present, the workflow ends. If inconsistencies are present, again the workflow starts.

   Our extension covers the following steps, to show the knowledge engineer how the relevant axiom leading to repair candidate(s) can be weakened.
7. The knowledge engineer decides to investigate the repair candidate further. This happens when the knowledge engineer selects the repair candidate and is taken to the screen where they can view the minimal conflict sets associated with the repair (refer to Section 2.6 for more information on minimal conflict sets).
8. To isolate the issue, a specific minimal conflict set needs to be selected to apply the next computations on. The extension highlights to the knowledge engineer the minimal conflict set most suited for weakening. It selects the minimal conflict set with the individual or generated individual that is more general/ higher up in the hierarchy

Stellenbosch University https://scholar.sun.ac.za

than the other individuals; if there is more than one conflict set which meets this criteria, it selects the minimal conflict sets that contains the fewest concepts. For a more detailed explanation of this algorithm, refer to Section 3.3. Importantly, the user will be able to override this choice of minimal conflict set, and select a different minimal conflict set to perform the next steps on.

9. Next, the extension computes the rank of all concepts, and an axiom is selected for weakening (refer to Section 3.2 for more details).
10. Then, the extension obtains the weakened axiom form (refer to Section 3.2 for more details).
11. This weakened result is presented to the knowledge engineer, and if they accept the weakened result, the ontology is updated, and again OntoDebug checks for inconsistencies or incoherences. If any are present, the process is repeated.

With the design of our solution being in the form of an extension, this means that inconsistencies or incoherences not caused by the exceptionality pattern this thesis is investigating can still be repaired as normal. For a detailed examination of the benefits and challenges of this approach, along with an evaluation of how the extension performs with a range of different situations, please refer to the next chapter.

3.5.2. Integration with initial repair-generation tasks: Not only does the OntoDebug tool lead the knowledge engineer through an interactive process to minimise the number of diagnoses returned, but it also gives the knowledge engineer the diagnoses (or repair candidates) up-front. If the diagnoses are not too extensive, the knowledge engineer may be able to more quickly pinpoint the fault. So there is a trade-off between *systematically* finding the repair candidates, and finding the repair candidates *easily/quickly* – the knowledge engineer chooses which to follow.

From the above, we've already seen how our extension slots into the interactive ontology debugging workflow. With reference to the more detailed investigation in Section 3.4, there are two ways in which weakened results can be posed as repairs for repair candidates:

1. Following a **model-based diagnosis approach**, at the time when the diagnoses are computed (see point 2 above), for each diagnosis, the relevant minimal conflict set can be selected by the extension (refer to Section 3.3). Then an axiom for weakening can be selected, and the weakened result (refer to Section 3.2) can be displayed to

Stellenbosch University https://scholar.sun.ac.za

the knowledge engineer. Chapter 5 discusses how future work could investigate the computation effort required for this.

2. Following a **heuristic approach**, exceptionality patterns (see Definition 22) can be identified, without running the reasoner, and axiomatic weakening as a design pattern can be followed to show repair suggestions to the user. Chapter 5 discusses how future work could investigate tighter integration between the model-based and heuristic approaches to ontology debugging.

We propose that a separate window be available in OntoDebug to accommodate suggestions from heuristic approaches to ontology debugging. Potentially in the OntoDebug configuration settings, users can also set whether they would want heuristic and model-based suggestions to be generated – please refer to Chapter 5 for further discussion on this.

3.6. Related work

Closely related our our work on effectuating a repair that is as minimal as possible is the work of Horridge et al. (2008): they argue that often in real-world ontologies, axioms are not in their minimal form, and contain 'long' axioms which can be broken down into smaller pieces. Let's continue with a different variant of our running example ontology:

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

It is clear that the fact that Staff have access to the $\mathsf{CanteenMenu}$ does not directly contribute to the incoherence in the above ontology. However, when justifications are computed, the third axiom as a whole will be returned in the justification. For this reason, Horridge et al. (2008) have developed a way of breaking down longer axioms into their short form, and then computing the justifications from there. The effect of this is that only the relevant parts of axioms leading to the incoherence are identified, and then only those parts are removed.

Stellenbosch University https://scholar.sun.ac.za

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

3.7. Summary

In this chapter, we have shown the following:

1. For all its merits, the OntoDebug methodology and tool can lead to unintuitive results particularly where we are dealing with faulty modelling due to axioms that are not necessarily incorrect, but that need to be weakened. This acts as a motivation for an extension to the OntoDebug tool, which can when prompted provide suggestions on how an axiom in the minimal conflict set can be weakened rather than deleted.
2. We have provided a methodological outline which describes how suggestions can be generated for weakening, rather than deleting, axioms. In the methodology, the issue is first isolated by selecting the minimal conflict leading to the diagnosis; then through using the ranking algorithm, we determine a candidate axiom to weaken and a candidate concept with which to weaken it. Weakening is applied using the postulate of Cautious Monotonicity. The weakened result is shown to the knowledge engineer and the knowledge engineer can choose to accept or reject the solution.
3. Through constructing the methodological outline, it became apparent that not only could our methodology be used as part of a model-based diagnosis approach to ontology debugging, but that it can be combined with a heuristic approach to ontology debugging as well. Especially for large ontologies where reasoning tasks may be expensive, this may assist with debugging tasks. This also shows that there is scope for combining model-based diagnosis and heuristic approaches in the debugging community.
4. Finally we described how this extension fits in with the larger OntoDebug workflow. In essence, it does not detract from the standard OntoDebug workflow, but does provide a way to query results further and to weaken (rather than delete) the faulty axiom. We also discuss how, during the task of initial repair-generation, we can apply our methodology either by following a model-based diagnosis or heuristic approach.

Stellenbosch University https://scholar.sun.ac.za

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.

Stellenbosch University https://scholar.sun.ac.za

Chapter 4

Evaluation and Discussion

In this chapter, debugging via weakening is critically assessed. First, an evaluation of edge-cases ensues and we investigate the behaviour of the debugging methodology in each of these cases. This first section gives a more technical discussion, and is intended to conceptually stress-test debugging via weakening, and provide insight on where future work may originate from.

Secondly, we assess debugging via weakening from a more high-level view. This section is intended to evaluate the debugging approach from a general view, and how it contributes to the debugging and defeasible DL communities.

4.1. Testing behaviour with edge-cases

By assessing the behaviour of debugging via weakening at edge-cases, prompts for future research may emerge, along with identifying where the boundaries of the methodology lie. Specifically, we look at the behaviour of the methodology in the following cases:

- Cases where multi-level exceptionality inconsistencies are **entangled** alongside inconsistencies/ incoherences caused by a different issue;
- Cases where, during ranking, a concept is assigned a **rank of** ∞;
- Cases where a concept is exceptional in one **context**, but not in another[2].

4.1.1. Entangled inconsistencies/incoherence: Entangled inconsistencies/ incoherence refer to instances where one concept is unsatisfiable due to more than one reason. For our purposes, we want to investigate where one of the inconsistencies/ incoherences is caused by an axiom that has been asserted too strongly, and that needs to be weakened. When inconsistencies are not entangled – that is to say when a multi-level exceptionality inconsistency appears, and the concepts in this conflict set are not entangled with other

[2] In this case we are not referring to formal reasoning with context as in Britz and Varzinczak (2019); we are rather referring to context in the general sense of the word.

Stellenbosch University https://scholar.sun.ac.za

conflict sets – it is plain to see that the debugging via weakening methodology would return the expected results as our methodology is simply an *extension* of the original OntoDebug methodology: there would simply be more than one diagnosis returned as the exceptions are unrelated.

It is not immediately intuitively clear whether this same posit holds for entangled inconsistencies/ incoherence. Let's investigate with an amalgamation of the examples used throughout Sections 2.7 and 3.2:

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

Let us assume that Staff has an individual associated with it. Staff is an unsatisfiable concept for two reasons in the above ontology: 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 OntoDebug methodology, we see that for the above example, two axioms are suggested as needing to be repaired, due to two minimal conflict sets being involved in causing the incoherence:

Stellenbosch University https://scholar.sun.ac.za

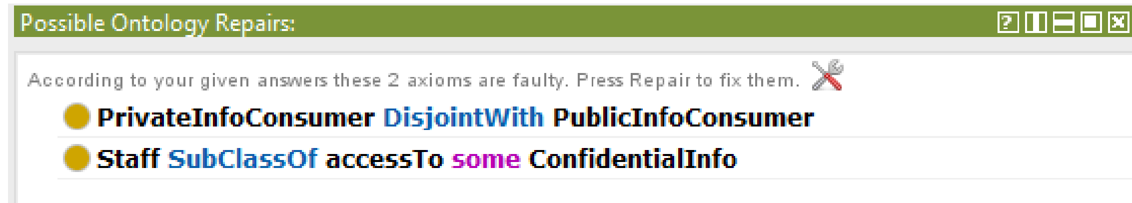


Figure 32 Running entangled concepts through OntoDebug returns separate axioms for repair.

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 (individuals, for instance, who are Staff should have access to ConfidentialInfo **in addition to** having access to PublicInfo). The minimal conflict set related to the second axiom returned as an axiom to repair can be solved following the methodology proposed in this thesis by weakening this statement: $\mathsf{User} \sqsubseteq \exists \mathsf{accessTo}.\mathsf{PublicInfo}$ – refer to Section 3.2 for a detailed breakdown of this methodology. In whichever order these are assessed and fixed, the results remain the same, and so we can assert that our extended debugging methodology works in cases where a concept is entangled in more than one conflict set.

4.1.2. Infinity rankings: Where we discuss our weakening methodology in Section 3.2, it is clear that ranking of the concepts by their level of exceptionality plays an important role. In Section 2.3, we also saw that at times the ranking of a concept can be ∞, which means that even when ordering concepts by their level of exceptionality, a concept is exceptional on its same level of exceptionality – ultimately, it is entirely unsatisfiable. Let's assess this with the following example:

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

In this example, the concept of Staff is not only exceptional to User, the level 0 concept, but also to itself, which leads it to have a ranking of ∞. In this case, the methodology

Stellenbosch University https://scholar.sun.ac.za

would give an error, as it only works with concepts at levels 0 and 1 and the user would need to manually investigate.

An important part of the debugging process is that it is *interactive*: it ought to be remembered that the debugging methodology is in place to guide the user, and that it likely can never be fully automated as human interpretation is required for domain knowledge that aligns with our view of the world to be accurately captured.

4.1.3. Context-bound exceptionality: Certain forms of reasoning (for instance Rational Closure) cannot deal with the presumption of independence. The presumption of independence states that if a concept is exceptional in one context, then it is not necessarily the case that it is exceptional in other contexts, and therefore inferences related to that concept should still assume that the concept is a 'normal' concept at level 0. We investigate whether our current solution can deal with the presumption of independence. Let's first take the following ontology as a test case:

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

In this example, after our solution is applied, the first axiom in the above ontology is changed to the following:

$\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubseteq \neg\exists \mathsf{accessTo.ConfidentialInfo}$

When this change is made, and the reasoner is run to find inferences, we find that although it is the case that in the context of having $\mathsf{accessTo.ConfidentialInfo}$, Staff acts in a different way to its parent, User, in the context of having $\mathsf{accessTo.PublicInfo}$, it is correctly inferred that the concept of Staff behaves in the same way as its parent, User, as this is an independent context. In this case then, the presumption of independence is maintained using this debugging methodology.

There are other examples, however, where the presumption of independence would not hold. Take for instance:

Stellenbosch University https://scholar.sun.ac.za

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

The knowledge engineer might want to assert the above to capture what sorts of users are system users/ bots vs normal human users. We would want the reasoner to infer that the concept Staff, being subsumed by User, and User not having accessTo.ConfidentialInfo is therefore subsumed by Human.

Using our repair strategy, again the repair would be the following:

$\mathsf{User} \sqcap \neg\mathsf{Staff} \sqsubseteq \neg\exists \mathsf{accessTo}.\mathsf{ConfidentialInfo}$

In this case, however, the reasoner would not be able to infer that Staff is subsumed by Human, and thus a potentially valuable piece of knowledge is missed out on.

Currently, this thesis focuses on one of the postulates in formulating a repair strategy, namely Cautious Monotonicity (CM). Other postulates may also provide the user with guidelines on possible repairs/ debugging. In this case, for instance, the Right Weakening (RW) postulate can be used:

$$\text{(RW)} \quad \frac{C \sqsubsetsim D, \ D \sqsubseteq E}{C \sqsubsetsim E}$$

When we substitute the first premise with the axiom related to the concept at level 0, and the second premise with the axiom using the first premise axiom (but that is outside the minimal conflict set), this postulate reads:

$$\text{(RW)} \quad \frac{\mathsf{User} \sqsubsetsim \neg\exists \mathsf{accessTo}.\mathsf{ConfidentialInfo}, \ \top \sqcap \neg\exists \mathsf{accessTo}.\mathsf{ConfidentialInfo} \sqsubseteq \mathsf{Human}}{\mathsf{User} \sqsubsetsim \mathsf{Human}}$$

In this case, more work will need to be done to establish the impact this postulate will have on the debugging methodology, and particular attention will need to be paid on how to identify the second premise which is outside of the minimal conflict set.

Stellenbosch University https://scholar.sun.ac.za

4.2. Contributions to debugging and defeasible DL communities

Carlile (2002) notes that innovation is defined as sourcing new knowledge at the intersection between two disciplines. This work has investigated how defeasible DL tools and concepts – such as ranking and the usage of postulates – can contribute to the ontology debugging community by providing the knowledge engineer with recommendations on how axioms can be weakened rather than deleted. For the defeasible DL community, this provides an immediately applicable use of defeasible DL tools and concepts to other sub-communities in the overarching ontology community. For the ontology debugging community, this provides a way of extending existing interactive ontology debugging tools so that not only axioms at the heart of the fault (diagnoses) are presented to the user, but so that concrete recommendations for fixing these faulty axioms is also provided. The next Chapter further explains how the ties between these two communities can be strengthened as a result of future research in this area. Suffice it to say that a new area for investigation at the intersection between the debugging and defeasible DL communities has been successfully opened.

The main contribution of this work is to enable 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. Our extension to interactive ontology debugging enables the knowledge engineer not only to see *which* axioms need to be fixed, but *how* these axioms can be fixed. The extension follows on from the traditional OntoDebug methodology from the point where the diagnoses are presented to the user: for each diagnosis, the concepts in the minimal conflict set are ranked; after the ranking, the axiom relevant for weakening is identified; finally the ranking is then used in conjunction with the postulate of Cautious Monotonocity to weaken the relevant axiom and present this result to the user.

From this main contribution, there have also been spin-off successes: firstly, in Section 3.4 we saw that the same principle as what is applied in the main methodology can be translated to a design pattern and used in conjunction with a heuristic rather than model-based approach. The design pattern has been defined, and it is suggested that this is added to a catalogue of design patterns available to the user in their debugging environment. Secondly, as noted in Section 4.1, in certain circumstances, our extension can cater for

Stellenbosch University https://scholar.sun.ac.za

the presumption of independence so that if a concept is exceptional in one context, it is not necessarily inferred that this concept is exceptional in all contexts. The next chapter will outline how future research on this front can progress to be of benefit to the ontology debugging and defeasible DL communities by further investigating postulates as design patterns.

Of course, our extension covers only a certain scope of debugging activities. There are some limitations even when combining our approach with the traditional interactive ontology debugging approach: firstly, and as mentioned in Section 4.1, with our extension if a ranking of ∞ is encountered, the concept is ultimately unsatisfiable, and a suggestion will not be generated. It is important to note that the onus of decision-making still lies with the knowledge engineer, and the methodology should only be used to guide the knowledge engineer in their understanding of the problem and the solution. Secondly, the core of our approach is based on a model-based diagnosis approach, with only a hint of using heuristics to find fault patterns that may represent multi-level inconsistencies. A model-based diagnosis approach will only pick up on inaccuracies in a knowledge base when there is an incoherence or inconsistency – even without any inconsistencies or incoherence there may still be a modelling inaccuracy leading to unfavourable results.

Stellenbosch University https://scholar.sun.ac.za

Chapter 5

Conclusion and Future Work

The benefit of ontologies lies in the fact that they 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 lies in (1) knowledge retention (2) without introducing undue logical inconsistencies.

As noted in Chapter 1, ontologies are likely to be subjected to the same 3Vs as big data: volume, velocity and variety. With ontologies constantly growing in size (**volume**), either by human or system users adding more concepts and relationships, inconsistencies *arise more often*. As the rate at which new concepts and relationships are added to an ontology (**velocity**) increases, either by multiple human and system users adding new concepts and relationships, or by merging two or more ontologies, these inconsistencies *arise faster* than previously. 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.

It has been argued that this presents a so-called *wicked problem*, and thus this problem is interesting to investigate from a design science perspective. This involved analysing current approaches to debugging, and creating 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. In the same way that Rodler (2015), Rodler et al. (2019) and Schekotihin et al. (2018) 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 feature showing *how* axioms could be weakened rather than deleted, valuable knowledge may be lost.

Stellenbosch University https://scholar.sun.ac.za

For each diagnosis, our extension suggests a way to fix the inconsistency/ incoherence 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 and incoherence are resolved. This same logic can also be applied using a more heuristic approach, which involves identifying multi-level exceptionality patterns, and applying systematic weakening as a design pattern.

Of the OntoDebug tool, (Rodler 2015, p. 30) states: "Note that, due to monotonicity of $\mathcal{L}$, only deletion (and not expansion) of the knowledge base can effectuate a repair of inconsistencies." With our extension, we update the above statement: "due to monotonicity of $\mathcal{L}$, deletion *or weakening of an axiom* (but not expansion) of the knowledge based can effectuate a repair of inconsistencies". It is in this way that 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.

5.1. Future Work

This thesis has put forward an extension to interactive ontology debugging that enables, through the use of defeasible reasoning tools, a way to suggest how faulty axioms can be repaired by weakening, rather than deleting, the faulty axioms. This work has been done at the design level, and in future would need to be implemented as a Protégé plug-in, as an extension to OntoDebug.

Stellenbosch University https://scholar.sun.ac.za

Certain algorithms playing a significant role in the development of this extension have already been implemented: Meyer et al. (2014) 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. Implementing the extension as a Protégé plug-in would therefore involve:

- **Determining how the OntoDebug and DIP plugins would integrate with each other.** Either a new, standalone plug-in, incorporating aspects from both would need to be created; or a plug-in which makes calls to each independent one of these plug-ins could be created. The former approach may be easier for the end-user to install and navigate, but the latter approach would mean that minimal rework would need to be done – i.e. if any bug fixes or enhancements need to be done, these are done in the original OntoDebug and DIP plug-ins, and these fixes/ enhancements would not need to be duplicated in the plug-in integrating the two. Another consideration would be to incorporate DIP directly in OntoDebug itself.
- **In the new plug-in, pull through the existing OntoDebug/ DIP algorithms required for the new extension.** From OntoDebug, all modules will be necessary to pull through (this could be a motivation for incorporating DIP directly into OntoDebug itself, rather than creating a separate extension). From DIP, get the algorithm that gets a concept's ranking.
- **Incorporate algorithms.** Using figure 31, apply new logic for when the user selects 'Investigate more' on a specific faulty axiom in a diagnosis. OntoDebug functionality would be followed up until this point; after this point, use the DIP algorithm for obtaining a concept's rank; then logic will need to be written to weaken the faulty classical axiom using Cautious Monotonicity. The program will then need to be able to replace the original axiom with the weakened axiom and to call back to the original OntoDebug tool again to perform a check for inconsistencies. This process is repeated as per the original OntoDebug logic.

Once implemented, computational analyses and user studies ought to be performed: computational analyses will evaluate how well, computationally, the tool performs; i.e. they investigate how well the tool scales along with the accuracy of the outputs. User studies would focus on whether a typical user with the relevant ontology building background has

Stellenbosch University https://scholar.sun.ac.za

the ability to use their judgement *effectively* in deciding whether a weakened axiom presents more accurately the domain they are modelling, or whether the faulty axiom should rather be deleted; user studies could also focus on the extent to which the OntoDebug extension helps a user to identify the correct repair in less time, and thus more *efficiently*, than if the extension were not present. A good starting point for a user study would be to follow a smiliar format as what is followed in Schekotihin et al. (2018): that is, given a faulty ontology a control group consisting of a statistically adequate number of users with relevant background in building ontologies use the previous ontology debugging tool (in this case, just OntoDebug without the extension), and metrics are captured on how effectively and efficiently the faults in the ontology are resolved. Meanwhile, the test group use the latest ontology debugging tool (in this case, OntoDebug, along with the extension), and the same metrics are captured for this group. The results are then compared, and it is determined whether the performance of users whilst using the OntoDebug along with its extension provides a statistically significant improvement in comparison to using just the OntoDebug tool.

Once it has been shown that the extension boosts users' efficiency and effectiveness when debugging ontologies, further work can be done to investigate how tools, algorithms and methodologies from the defeasible DL community can be used in the ontology debugging community, and *vice versa*. For example, more research can be performed to determine whether other postulates provide useful design patterns for heuristic ontology debugging: in this thesis, we have seen that Cautious Monotonicity and Right Weakening represent design patterns that crop up when debugging ontologies, yet there are a myriad other postulates (see Definition 5) which could also be inspected for similar gains. There are also some even more nuanced forms of defeasible reasoning that are being investigated: contextual defeasibility, for instance, as put forward by Britz and Varzinczak (2019). Emerging forms of defeasible reasoning may further assist the ontology debugging community with providing users with concrete, yet nuanced ways of solving conflicts.

Through a more ontology debugging focused lens, future work could also investigate how the model-based diagnosis and the heuristics approaches to ontology debugging could be more seamlessly combined. In this thesis, we have touched upon it, with our suggestion of building in the capability to recognise a design pattern that leads to multi-level exceptionality, and combining this with the model-based approach in OntoDebug. Future work

Stellenbosch University https://scholar.sun.ac.za

could focus on how other exceptionality patterns could be formalised as design patterns to be used in a heuristic approach alongside a model-based diagnosis approach. Especially for large ontologies, the benefit of a heuristic approach is that the reasoner does not need to be run to pick up on these design patterns – a design pattern could even be picked up on whilst the knowledge engineer is creating or modifying an ontology. Of course, a model-based approach which finds the faulty axioms first, and then suggest a repair, will return more accurate results. It may be possible that performance gains – both in terms of system and user performance – may improve if, whilst modifying a large ontology, a heuristic approach is followed to pinpoint any potential faults as they arise, and then to follow a model-based diagnosis approach when running the reasoner and checking for inconsistencies.

Finally, future work could also focus on how OntoDebug and the extension put forward in this thesis can form part of a larger ontology management tool. For instance, whether it can form part of the Optique tool proposed by Haase et al. (2013), which extracts an initial ontology and mappings from data sources, performs mapping and alignment, checks ontologies for defects, and provides versioning support. The purpose of this full package of tools is to enable access to relevant 'big' data by bringing together data from different sources through using an ontology and mappings. It is in the checking and corrections of defects that OntoDebug and its extension can be of use.

Stellenbosch University https://scholar.sun.ac.za

List of Figures

1 Visualisation of the SNOMED CT ontology. Black nodes represent concepts; grey lines represent relationships between the concepts. 2
2 Example of class hierarchy in Protégé. 27
3 Subsumption statement associated with User. 27
4 Subsumption statement associated with Staff. 28
5 Subsumption statement associated with BlackListedStaff. 28
6 Axioms noted to be strict/ classical axioms. 28
7 Axioms noted to be defeasible axioms. 29
8 Ranking in DIP in Protégé 5.0. 29
9 Input ontology used for illustration of OntoDebug. 42
10 Setting background knowledge in OntoDebug. 42
11 OntoDebug repair candidates – axioms which would, when deleted, effectuate a repair. 43
12 OntoDebug queries (first iteration). 43
13 In OntoDebug submitted answers become test cases; for each query answered at least one repair candidate is eliminated; if more than one repair still exists, further queries are posed to the knowledge engineer. 44
14 Iterative nature of OntoDebug. 44
15 Following a process of iteration, a minimal repair candidate has been identified. 45
16 OntoDebug's repair screen. 45
17 OntoDebug – repair success screen. 46
18 Ontology producing unintuitive result is created in Protégé 5.0. 48
19 Iteration 1 of list of repair candidates is given for ontology producing unintuitive result. 49
20 Iteration 1 queries generated of the ontology producing the unintuitive result. 49
21 Iteration 2 possible repair candidates to the ontology producing the unintuitive result. 50
22 Iteration 2 query answering to the ontology producing unintuitive results. . 50
23 Unintuitive repair candidate is provided. 51

Stellenbosch University https://scholar.sun.ac.za

24 Unintuitive axiom to repair. 60
25 List of justifications associated with unintuitive repair candidate. 61
26 OntoDebug behaviour when adding additional level of exceptionality. 63
27 OntoDebug behaviour when adding additional level of exceptionality: two repair candidates are listed. 63
28 Recurrent ontology design problem. 66
29 Ontology design pattern/ solution. 67
30 Iteration 1 of list of repairs is given for ontology producing unintuitive result. 68
31 Activity diagram: extension to OntoDebug – green boxes and lines represent proposed OntoDebug extension. 70
32 Running entangled concepts through OntoDebug returns separate axioms for repair. 78

List of Tables

1 Example of ranking output. 25
2 First iteration concept ranking output. 55
3 First iteration axiom ranking output. 55
4 Second iteration concept ranking output. 58
5 Second iteration axiom ranking output. 58

Stellenbosch University https://scholar.sun.ac.za

References

Aminu, E. F., Oyefolahan, I. O., Abdullahi, M. B. and Salaudeen, M. T. (2020), 'A review on ontology development methodologies for developing ontological knowledge representation systems for various domains.', *International Journal of Information Engineering & Electronic Business* **12**(2), 28–39.

Baader, F., Horrocks, I., Lutz, C. and Sattler, U. (2017), *Introduction to description logic*, Cambridge University Press.

Baader, F., Horrocks, I. and Sattler, U. (2004), Description logics, *in* S. Staab and R. Studer, eds, 'Handbook On Ontologies', Springer-Verlag, Berlin, chapter 1, pp. 3–28.

Baader, F. and Nutt, W. (2006), Basic description logics, *in* D. Nardi and R. Brachman, eds, 'Description Logic Handbook', Springer-Verlag, Berlin, chapter 2, pp. 47–100.

Banik, A. and Bandyopadhyay, S. (2016), 'Big data - a review on analysing 3Vs', *Journal of Scientific and Engineering Research* **3(1)**, 2394–2630.

Behrens, S. and Sedera, W. (2004), 'Why do shadow systems exist after an ERP implementation? Lessons from a case study', *PACIS 2004 Proceedings* p. 136.

Boutilier, C. (1994), 'Conditional logics of normality: A modal approach', *Artificial Intelligence* **68**, 87–154.

Britz, K., Casini, G., Meyer, T., Moodley, K. and Sattler, U. (2017), Rational defeasible reasoning for description logics, Technical report, University of Cape Town, South Africa.

Britz, K., Casini, G., Meyer, T., Moodley, K., Sattler, U. and Varzinczak, I. (2020), 'Principles of KLM-style defeasible description logics', *ACM Transactions on Computational Logic (TOCL)* **22**(1), 1–46.

Britz, K., Casini, G., Meyer, T. and Varzinczak, I. (2019), A KLM perspective on defeasible reasoning for description logics, *in* C. Lutz, U. Sattler, C. Tinelli, A. Turhan and F. Wolter, eds, 'Description Logic, Theory Combination, and All That: Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday', Springer International Publishing, Cham, pp. 147–173.
URL: *https://doi.org/10.1007/978-3-030-22102-7$_7$*

Britz, K. and Varzinczak, I. (2019), 'Contextual rational closure for defeasible $\mathcal{ALC}$', *Annals of Mathematics and Artificial Intelligence* **87**(1-2), 83–108.

Carlile, P. R. (2002), 'A pragmatic view of knowledge and boundaries: Boundary objects in new product development', *Organization Science* **13**, 442–455.

Casini, G., Meyer, T., Moodley, K. and I.Varzinczak (2013), Towards practical defeasible reasoning for description logics, 26th International Workshop on Description Logics.

Chan, P. and Hankel, L. (2019), System for detecting data protection violations, *in* 'ICCWS 2019 14th International Conference on Cyber Warfare and Security: ICCWS 2019', Academic Conferences and publishing limited, p. 30.

Stellenbosch University https://scholar.sun.ac.za

Friedrich, G. and Schekotykhin, K. (2005), A general diagnosis method for ontologies, Proceedings of the 4th International Semantic Web Conference (ISWC 2005).

Gangemi, A. and Presutti, V. (2009), Ontology design patterns, *in* 'Handbook on ontologies', Springer, pp. 221–243.

Giordano, L., Gliozzi, V., Olivetti, N. and Pozzato, G. L. (2013), 'A non-monotonic description logic for reasoning about typicality', *Artificial Intelligence* **195**, 165–202.

Guarino, N. (1995), 'Formal ontology, conceptual analysis and knowledge representation', *Knowledge Acquisition* **2**(3), 241–258.

Haase, P., Horrocks, I., Hovland, D., Hubauer, T., Jiménez, E., Kharlamov, E., Klüwer, J., Pinkel, C., Rosati, R., Santarelli, V., Soylu, A. and Zheleznyakov, D. (2013), 'Optique system: towards ontology and mapping management in OBDA solutions'.

Hevner, A. and Chatterjee, S. (2010), Design science research in information systems, *in* 'Design research in information systems', Springer, pp. 9–22.

Hevner, A. R., March, S. T., Park, J. and Ram, S. (2004), 'Design science in information systems research', *MIS quarterly* pp. 75–105.

Horridge, M., Bail, S., Parsia, B. and Sattler, U. (2011), The cognitive complexity of OWL justifications, *in* 'International Semantic Web Conference', Springer, pp. 241–256.

Horridge, M., Bail, S., Parsia, B. and Sattler, U. (2013), 'Toward cognitive support for OWL justifications', *Knowledge-Based Systems* **53**, 66–79.

Horridge, M., Parsia, B. and Sattler, U. (2008), Laconic and precise justifications in OWL, *in* A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin and K. Thirunarayan, eds, 'The Semantic Web - ISWC 2008', Springer Berlin Heidelberg.

Kalyanpur, A., Parsia, B., Sirin, E. and Cuenca-Grau, B. (2006), Repairing unsatisfiable concepts in OWL ontologies, *in* 'European Semantic Web Conference', Springer, pp. 170–184.

Knorr, M., Hitzler, P. and Maier, F. (2012), 'Reconciling OWL and non-monotonic rules for the semantic web'.

Kraus, S., Lehmann, D. and Magidor, M. (1990), 'Nonmonotonic reasoning, preferential models and cumulative logics', *Artificial Intelligence* **44**, 167–207.

Lambrix, P. (2019), 'Completing and debugging ontologies: state of the art and challenges', *ArXiv* **abs/1908.03171**.

McDermott, D. and Doyle, J. (1980), 'Non-monotonic logic I', *Artificial intelligence* **13**(1-2), 41–72.

Meyer, T., Moodley, K. and Sattler, U. (2014), DIP: A defeasible-inference platform for owl ontologies, CEUR Workshop Proceedings.

Stellenbosch University https://scholar.sun.ac.za

Nardi, D. and Brachman, R. J. (2003), An introduction to description logics, *in* F. Baader, D. McGuinness, D. Nardi and P. Patel-Schneider, eds, 'The Description Logic Handbook: Theory, Implementation and Applications', Cambridge University Press, London, chapter 1, pp. 1–43.

Obrst, L. (2003), Ontologies for semantically interoperable systems, *in* 'Proceedings of the twelfth international conference on information and knowledge management', pp. 366–369.

Peñaloza, R. (2019), Explaining axiom pinpointing, *in* 'Description Logic, Theory Combination, and All That', Springer, pp. 475–496.

Regulator, S. I. (2013), 'Protection of personal information act'.

Rodler, P. (2015), Interactive Debugging of Knowledge Bases, PhD thesis, Alpen-Adria University Klagenfurt.

Rodler, P., Jannach, D., Schekotihin, K. and Fleiss, P. (2019), 'Are query-based ontology debuggers really helping knowledge engineers?', *Knowledge-Based Systems* **179**, 92–107.

Rodler, P. and Schmid, W. (2018), On the impact and proper use of heuristics in test-driven ontology debugging, *in* 'International Joint Conference on Rules and Reasoning', Springer, pp. 164–184.

Rosnizam, M. R. A. B., Kee, D. M. H., Akhir, M. E. H. B. M., Shahqira, M., Yusoff, M. A. H. B. M., Budiman, R. S. and Alajmi, A. M. (2020), 'Market opportunities and challenges: A case study of Tesco', *Journal of the community development in Asia* **3**(2), 18–27.

Roussey, C., Pinet, F., Kang, M. and Corcho, O. (2011), An introduction to ontologies and ontology engineering, *in* G. Falquet, C. Metral, J. Teller and C. Tweed, eds, 'Ontologies in Urban Development Projects', Springer, London, chapter 2, pp. 9–38.

Schekotihin, K., Rodler, P. and Schmid, W. (2018), Ontodebug: interactive ontology debugging plug-in for Protégé, *in* 'International Symposium on Foundations of Information and Knowledge Systems', Springer, pp. 340–359.

Schlobach, S., Huang, Z., R.Cornet and Harmelen, F. (2007), 'Debugging incoherent terminologies', *Journal of automated reasoning* **39**, 317–349.

Shoham, Y. (1987), Nonmonotonic logics: Meaning and utility., *in* 'IJCAI', Vol. 10, Citeseer, pp. 388–393.

Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D. and Horrocks, I. (2013), OptiqueVQS: Towards an ontology-based visual query system for big data, *in* 'Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems', MEDES '13, Association for Computing Machinery, New York, NY, USA, p. 119–126.
URL: *https://doi.org/10.1145/2536146.2536149*

Studer, R., Benjamins, V. R. and Fensel, D. (1998), 'Knowledge engineering: Principles and methods', *Data and Knowledge Engineering* **25**, 161–197.

Troquard, N., Confalonieri, R., Galliani, P., Penaloza, R., Porello, D. and Kutz, O. (2018), Repairing ontologies via axiom weakening, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 32.

Stellenbosch University https://scholar.sun.ac.za

Tziva, M., Negro, S., Kalfagianni, A. and Hekkert, M. (2019), 'Understanding the protein transition: the rise of plant-based meat substitutes', *Environmental Innovation and Societal Transitions* .

Varzinczak, I. (2018), 'A note on a description logic of concept and role typicality for defeasible reasoning over ontologies', *Logica Universalis* **12**(3-4), 297–325.

Wang, H., Horridge, M., Rector, A., Drummond, N. and Seidenberg, J. (2005), Debugging OWL-DL ontologies: A heuristic approach, *in* 'International Semantic Web Conference', Springer, pp. 745–757.

Yamada, N. and Fukuta, N. (2016), Toward performance-oriented ontology debugging support using heuristic approaches and dl reasoning, *in* '2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)', IEEE, pp. 88–91.