Methods for Resolving Inconsistencies in Ontologies

Sik Chun Joey Lam

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
of the
University of Aberdeen.

Department of Computing Science

June 20, 2007

Declaration

Abstract

OWL ontologies are becoming increasingly important to the Semantic Web. Inconsistencies in OWL ontologies can easily occur, and it is a challenging task for ontology modellers, especially for non-expert ontology modellers, to resolve such inconsistencies. Existing ontology debugging tools can identify the problematic axioms for unsatisfiable concepts. However, current systems provide limited support for selecting which problematic axioms to remove/modify to resolve the unsatisfiability. In the case of modification, the modellers are provided with limited information about the impact of changes on the ontology. Therefore, there is a growing need for tools that can provide guidance on (1) selecting the axioms to modify, and (2) how to minimise the impact of proposed modifications on the ontology.

We firstly extend an existing tableau-tracing technique for debugging in order to pinpoint problematic *parts* of axioms, which are responsible for the unsatisfiability of a concept. This information allows us to rewrite faulty axioms by removing error-causing concepts, instead of removing axioms entirely. Secondly, we investigate the *heuristics* used by ontology engineers to resolve inconsistencies in ontologies. A number of empirical studies have been conducted to acquire a variety of such heuristics from ontology engineers. Some of the acquired heuristics are already incorporated in existing tools, however, we have also discovered a sizeable number of additional useful heuristics. Moreover, we investigate the usefulness of incorporating these heuristics in a software tool to guide non-expert ontology modellers to select appropriate axioms for modification. The result of our usability study shows that the heuristics are useful to help ontology users debug ontologies. Thirdly, in the case of modification, we study the usefulness of providing the modeller with detailed information about the *impact of changes* on the ontology. Specifically, we utilise the extended tableau-tracing technique to identify changes which are helpful in that they restore lost entailments (due to axiom removal), and those that are harmful in that they cause additional unsatisfiability.

The work presented in this thesis constitutes an advance in debugging ontologies. The thesis combines a heuristic approach and formal methods for dealing with inconsistencies in ontologies. It contributes to knowledge in that it shows that the approach of acquiring human heuristics and encoding them in a tool is viable for ontology debugging. It also shows that the fine-grained approach to indicating lost entailments and recommending modifications is useful. The results of this thesis should be valuable to both the implementors of ontology management tools, and to ontology users, especially for the less experienced users of ontologies.

Acknowledgements

I would like to thank my supervisors, Derek Sleeman, Wamberto Vasconcelos and Jeff Z. Pan, for their support throughout the long year of this PhD. Without their assistance the work embodied in the thesis would not have been possible.

I would like to thank many people in the Third Summer School on Ontological Engineering and the Semantic Web (SSSW'05), members of the Aberdeen AKT Research Group, as well as the students in the Computing Science Department at the University of Aberdeen. Without their participation in the studies the thesis would not have been possible. I also especially thank Alan Rector, Tim Norman and Denny Vrandečić for providing stimulating discussions and comments.

Most importantly, a particular thank you goes to my parents, Pui Chun, Ivy and Hung, for providing love and support over all of these years, and for making me feel part of the family even though I am miles away.

This work was supported under the EPSRC's grant number GR/N15764 and the Advanced Knowledge Technologies Interdisciplinary Research Collaboration, which comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Semantic Web and Ontologies

The Semantic Web [19; 25] is an extension of the current World Wide Web, which enables the accessing of Web resources by semantic content rather than just by keywords. Descriptions of Web resources (i.e., Web pages and a wide range of Web accessible data and resources) are structured and made at least partially machine-understandable, so that information can be exchanged and processed by both automated agents and humans. These descriptions of Web resources are usually defined by one or more ontologies. Ontologies [56; 147] play a major role in supporting the Semantic Web because they can help in the representation of the content of Web resources in a formal manner, thus facilitating information sharing and reuse. Typically, an ontology consists of a finite list of terms and the relationships between these terms. The *terms* denote important domain concepts (classes of objects). The *relations* denote binary relationships between two terms.

Example 1 *For example, in an animal domain,* animals, lions, sheep, *and* cows *are some of the important concepts. We can relate these terms with each other, i.e.,* lions eat animals, *where* eat *is a binary relationship.*

The Resource Description Framework (RDF) [107] is a language for representing information about resources in the Web. It provides a metadata scheme that can be used to describe Web resources. The fundamental concepts of RDF are resources, properties and statements:

- Resources are any type of object described by RDF. Every resource has a URI, a Universal Resource Identifier. A URI can be a URL (Unified Resource Locator) or some other kind of unique identifier.

- Properties (also known as predicates) define attributes or relations used to describe a resource. For example, eat, hasName. Properties in RDF are also identified by URIs.

- Statements assign a value to a property in a specific resource. Just as an English sentence usually comprises of a subject, a verb and object, RDF statements consist of subjects, properties and objects. For example, lions eat animals. If we represent this sentence in RDF, lions and animals are resources, denoted graphically by nodes, while eat is a property, denoted graphically by an edge.

Example 2 *An example of RDF statement,* Peter is the creator of the resource http://www.example.org/∼peter. *This statement is presented in a directed labelled graph in Figure 1.1.*

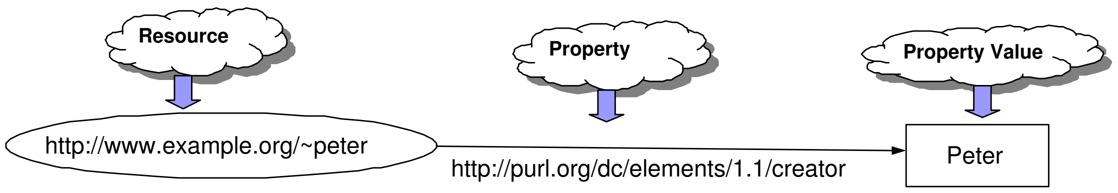

Resource
Property
Property Value
http://www.example.org/~peter
http://purl.org/dc/elements/1.1/creator
Peter

Figure 1.1: An RDF Graph

It is important to note that the RDF data model does not provide modelling primitives for defining the relationships between properties and resources. For example, in RDF we cannot specify that the relation eat can only be used between instances of the classes lions and animals. This limitation is solved by the RDF Vocabulary Description Language [23], also known as RDF Schema or RDFS. RDFS is a recommendation of the W3C that extends RDF with frame-based primitives. The combination of RDF and RDF Schema is written as RDF(S).

RDF(S) represent a first step forward producing semantic markups, they are useful for expressing taxonomies and inherited relationships; these relationships are not expressed directly in the structure of the document. However, the expressivity of RDF(S) is very limited [5]: RDF is (roughly) limited to binary ground predicates, and RDFS is (roughly) limited to a subclass hierarchy and property hierarchy, with domain and range definitions of properties. In particular, we have no standard way in RDF for expressing disjointedness, unions and intersections of particular classes, equivalence between classes etc. For example, RDF/RDFS provides no standard definitions for directly expressing that lions are not mammals, a human is either female or male. On the other hand, the Web Ontology Working Group of W3C identified the need for a more expressive ontology modelling language than RDF(S). This shows that efficient reasoning support and sufficient expressive power are important for ontology languages. To fulfil these requirements, the Web Ontology Language (OWL) [18] was released as a W3C recommendation in 2004. OWL is built upon RDF and RDFS for processing information on the Web; further it supports automated reasoning. OWL provides constructs to create new class descriptions as logical combinations (intersections, unions, or complements) of other classes, define value and cardinality restrictions on properties, and so on.

Example 3 *Using OWL, we can define that the domain of* eat *is* animals, lions eat something which is the union of sheep and cows, lions only eat animals*,* sheep *are disjoint with* cows.

A major feature of OWL is that it supports various types of inference; typically, consistency (i.e., no contradictory information) and subsumption relationships (i.e., a particular class is a subclass of another) etc. Further, it has a formal semantics based on Description Logics (DLs) [7]. Note that the statements asserted about an ontology and formulated in DL syntax are called *axioms*.

Example 4 *Continuing with Example 3, as the domain of* eat *is* animals*, and* lions eat at least an instance of animals*, we can infer that* all lions are animals.

This implicit information can be automatically inferred by a DL reasoner. Popular existing DL reasoners in the OWL community include FaCT [75], Pellet [136], FaCT++ [146] and RACER [60].

1.2 Inconsistencies in Ontologies

The importance of ontologies is not just limited to the Semantic Web, but also other applications, such as data integration [138], e-learning [144], life sciences [129], as well as medical and bioinformatics terminology systems [123; 17]. There are many organisations currently developing ontologies. Even for expert ontology engineers, support for building high quality ontologies is important, particularly in modelling complicated or moderately large ontologies. There are several OWL ontology editors, such as Protégé [117], KAON [120] and SWOOP [92], which support the design and construction of OWL ontologies. Still, inconsistencies or unintended implicit consequences of modelling can easily occur. We list below several typical scenarios which may cause inconsistencies in ontologies [132]:

- Misunderstanding OWL modelling – OWL is an expressive DL. Often, the domain experts themselves are the ontology designers. In this case, the designer of an ontology is not an expert in the ontology language used. Most such designers find it difficult to understand the logical meaning of the underlying description logic, and potential inferences in ontologies; hence people may fail to formulate axioms which are logically correct, or may specify contradictory statements. Rector et al. [125] list a number of common errors in modelling OWL ontologies, such as the confusion of 'some not' and 'not some', or misunderstanding the difference between the linguistic and logical use of 'and' and 'or'.

  Example 5 *For example*[1], *one has to assert that* lions eat cows and sheep, *which is correct from the viewpoint of linguistic usage. Some people will assert it as* lions $\sqsubseteq \exists$eat.(sheep $\sqcap$cow) *which actually means* lions eat something which is both sheep and cow, *this assertion causes a contradiction, because nothing can be both a* cow *and* sheep *simultaneously (*cows *are disjoint with* sheep*). The correct assertion should be* lions $\sqsubseteq \exists$eat.(sheep $\sqcup$cow) *which means* lions eat something which is either sheep or cow.

- Reusing/integrating ontologies – As the number of ontologies available on the Web is increasing, it is common for modellers to reuse other people's ontologies, or integrate parts from several existing ontologies. However, online ontologies are made for different purposes, in different contexts, and hence reusing/integrating ontologies may easily result in inconsistencies.

  Example 6 *A typical bird-penguin example: suppose there is an existing ontology on the Web, which contains (1)* birds are animals and can fly. *Assume someone downloads it and extends it by adding more statements, such as (2)* eagles are birds, *(3)* penguins are

[1]This example is borrowed from OilEd tutorial at http://oiled.man.ac.uk/tutorial/

birds and cannot fly. *As a result,* penguins *contains contradictory information, because* penguins *are* birds *which can* fly*, but* penguins *themselves cannot* fly.

- Migration to OWL – Migrating some data sources (e.g., KIF, DAML, frame-based knowledge source) which are represented in non-OWL representational formalism to OWL ontologies might result in inconsistencies because of the difference in the formalisms. For example, the DICE terminology (a DL terminology in the Intensive Care domain) suffered from a high number of unsatisfiable concepts due to its migration from a frame-based system [131].

1.3 Approaches to Debugging Ontologies

Existing DL reasoners can infer implicit information and detect which classes in the ontology contain contradictory information. By definition, an unsatisfiable class cannot have any individual or instance. An ontology containing unsatisfiable named classes is called *unsatisfiable*. An ontology is inconsistent iff it has no model. Inconsistent ontologies are those which have a contradiction in the individual data, e.g., an instance of an unsatisfiable class. For the bird-penguin example, the modeller is informed that Penguin is unsatisfiable, it cannot have any possible instance. However, traditional DL reasoners such as FaCT [75], cannot explain why the contradiction exists; they do not provide information on which axioms are contradictory. To address this limitation, there are existing approaches: (1) pinpointing axioms in an ontology which are responsible for an unsatisfiable concept [131] by interacting with an external DL reasoner; a set of minimally unsatisfiable sub-ontologies is obtained. The technique is particularly useful for large ontologies or ontologies with numerous unsatisfiable classes; (2) removing just enough axioms to obtain a set of maximally satisfiable subontologies [112]. This is achieved by a specialised tableau-like algorithm which calculates the maximally satisfiable subontologies directly, without relying on an external DL reasoner. The following example illustrates the two approaches.

Example 7 *In the bird-penguin example, using the first approach, two axioms are identified to be responsible for the unsatisfiable class, i.e.,*

(1) Bird $\sqsubseteq$ Animal $\sqcap$ CanFly [birds are animals and can fly]
(3) Penguin $\sqsubseteq$ Bird $\sqcap\neg$ CanFly [penguins are birds and cannot fly]
Removing or modifying any one of these axioms can resolve the unsatisfiability.

Using the second approach, two maximally satisfiable subontologies are obtained:

(1) Bird $\sqsubseteq$ Animal $\sqcap$ CanFly [birds are animals and can fly]
(2) Eagle $\sqsubseteq$ Bird [eagles are birds]
or
(2) Eagle $\sqsubseteq$ Bird [eagles are birds]
(3) Penguin $\sqsubseteq$ Bird $\sqcap\neg$ CanFly [penguins are birds and cannot fly]

Actually the two approaches give the same result. A maximally satisfiable subontology can be obtained by removing any one of axioms which is pinpointed by the first approach.

However, it is easy to see that we do not need to remove either the whole axiom (1) or (3) to resolve the unsatisfiability. In order to minimise the loss of information from the ontology, we can simply remove parts of the axioms, i.e., either remove the part birds can fly from axiom (1), or remove the part penguins are birds from axiom (3), and the ontology then becomes satisfiable.

We conclude that it is important to identify which parts of axioms are causing concept unsatisfiability, rather than the whole axioms. Axioms in OWL ontologies can be quite complicated, involving intersection or unions of concepts, but typically only parts of axioms actually cause the error. Therefore, ontology debugging tools should pinpoint the problematic parts of axioms, instead of the whole axioms.

After pinpointing the (parts of) axioms responsible for an unsatisfiable concept, the next step is to resolve the error by removing either one of the pinpointed axioms. It is difficult for tools to make specific suggestions for how to resolve problems, because it requires an understanding of the meaning of the represented classes. Only few existing approaches provide support for resolving inconsistencies in ontologies. Tools such as SWOOP [92] that provide this functionality are quite limited in their approach; the problematic axioms are ranked in order of *importance*. We use the animal example to illustrate their approach.

Example 8 *Continuing with the animal example, we add more axioms to the ontology. The ontology now contains the following axioms:*

(1) cows $\sqsubseteq$animals	[cows are a type of animal]*;*
(2) sheep $\sqsubseteq$animals	[sheep are a type of animal]*;*
(3) lions $\sqsubseteq \exists$ eat.(cows $\sqcap$ sheep)	[lions eat something which is both cow and sheep]*;*
(4) tigers $\sqsubseteq \exists$ eat.(cows $\sqcap$ sheep)	[tigers eat something which is both cow and sheep]*;*
(5) cows $\sqsubseteq \neg$ sheep	[cows and sheep are disjoint with each other]

A DL reasoner detects that both lions *and* tigers *are unsatisfiable. Axioms (3) and (5) cause* lions *to be unsatisfiable; axioms (4) and (5) cause* tigers *to be unsatisfiable. In this case, axiom (5) causes two unsatisfiable classes; its removal can resolve two unsatisfiabilities. Thus, axiom (5) is regarded as less important than axioms (3) and (4). Moreover, removing axiom (5) is a change which preserves more of the information in the ontology, as otherwise two axioms would have to be removed (axioms (3) and (4)). By removing axiom (5) we can retain the information of* lions *and* tigers*.* SWOOP *uses both of these heuristics, i.e., number of unsatisfiabilities caused by the axiom, and information lost by a modification. As a result,* SWOOP *will suggest that the user remove/modify axiom (5).*

Currently there is no consensus on how the “loss of information” can be measured [42]. In the above example, SWOOP calculates the loss of information in terms of the number and importance of the axioms that need to be removed from the ontology. We will describe the notion of information lost in Chapter 6.

Furthermore, it is obviously important to minimise the loss of information from the ontology due to changes. Intuitively, an axiom whose removal causes more information to be lost is regarded as more important, and should be preserved. However, it does not necessarily follow that such suggestions are the best results for all ontologies. A human ontology engineer may well propose a better solution in particular cases. For example, a human may believe that sibling classes are usually disjoint with each other. That means, the disjointness of cows and sheep should be kept in the above example. This heuristic has an opposite result to the factor of information lost.

Note that it is very difficult for tools to determine the optimal way to resolve such errors (semi)-automatically. Hence many tools simply leave it up to the user to determine the optimal way to resolve such errors. We believe that it will be useful to analyse the heuristics humans use to solve these kinds of problems, and to incorporate these heuristics in an ontology management tool. In this thesis, we describe an empirical study to acquire the heuristics used by ontology engineers when facing unsatisfiable ontologies (in Chapter 5). We aim to model the expertise and experience of the ontology engineers in order to give advice about which axioms to remove/modify.

Furthermore, whenever (parts of) an axiom is removed, it will be easy for ontology modelers to unintentionally remove implicit or intended information from the ontology. In order to minimise the impact on the ontology, it is important to calculate the lost entailments (e.g., subsumption relationships) of named concepts which occur due to the removal of (parts of) axioms.

Example 9 *For the bird-penguin example, the problematic axioms are:*
(1) $\mathsf{Bird} \sqsubseteq \mathsf{Animal} \sqcap \mathsf{CanFly}$ [birds are animals and can fly]*;*
(3) $\mathsf{Penguin} \sqsubseteq \mathsf{Bird} \sqcap \neg\,\mathsf{CanFly}$ [penguins are birds and cannot fly]

To resolve this problem, if we remove the part $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ *from axiom (1), then the implicit information* $\mathsf{Eagle} \sqsubseteq \mathsf{CanFly}$ *will be lost (see Figure 1.2 on the left-hand side). If we remove the part* $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ *from axiom (3), then the implicit information* $\mathsf{Penguin} \sqsubseteq \mathsf{Animal}$ *will be lost (see Figure 1.2 on the right-hand side).*

To minimise the loss of information, when $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ *is removed from axiom (1), the modeller should be notified that the information* $\mathsf{Eagle} \sqsubseteq \mathsf{CanFly}$ *should be added back to the ontology. When* $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ *is removed from axiom (3), the modeller should be notified that the information* $\mathsf{Penguin} \sqsubseteq \mathsf{Animal}$ *should be added back to the ontology.*

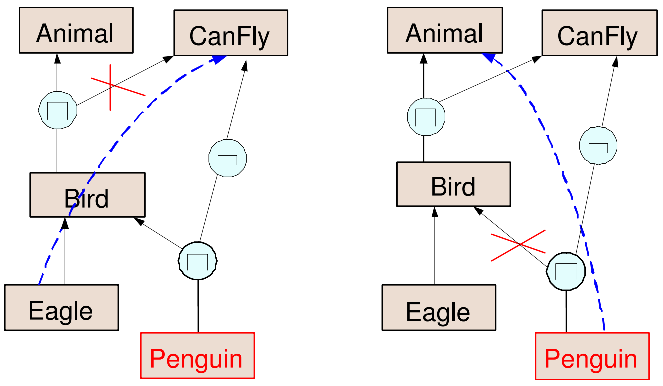


Figure 1.2: (Left-hand side) Option one: removing ‘birds can fly’, the implicit information ‘eagles can fly’ is lost (dashed arrow). (Right-hand side) Option two: removing ‘penguins are birds’, the implicit information ‘penguins are animals’ is lost. (NB: Deletion is indicated by a $\times$ on the appropriate link)

Whenever parts of an axiom or a whole axiom are removed, it frequently happens that intended entailments are lost. In order to minimise the impact on the ontology, we analyse the lost information (e.g., concept subsumption) of concepts due to the removal of (parts of) axioms. We also propose a fine-grained approach which allows us to identify changes which are *helpful* in that they restore lost information due to removal of axioms, and those that are *harmful* in that they cause additional unsatisfiability (in Chapter 7).

Example 10 *Continuing the bird-penguin example, when* $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ *from axiom (1) is removed, the helpful change is* $\mathsf{Eagle} \sqsubseteq \mathsf{CanFly}$*; when* $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ *from axiom (3) is removed, the helpful change is* $\mathsf{Penguin} \sqsubseteq \mathsf{Animal}$*. Harmful changes to replace the part* CanFly *in axiom (1) are* $\neg\mathsf{Animal}$, $\mathsf{Penguin}$, and Eagle*. This is because if the part* CanFly *in axiom (1) is replaced by* $\neg\mathsf{Animal}$*, then* Bird *will become unsatisfiable. If that part is replaced by* $\mathsf{Penguin}$*, then* $\mathsf{Penguin}$ *is defined as a type of* $\mathsf{Penguin}$*. If it is replaced by* Eagle*, then* $\mathsf{Penguin}$ *is defined as a type of* Eagle*.*

1.4 Contributions

This thesis makes the following contributions:

1. We extend previous tableau-tracing techniques to pinpoint problematic parts of axioms which are responsible for an unsatisfiable concept. This is a *fine-grained approach* which allows us to rewrite faulty axioms by removing error-causing concepts, instead of removing axioms entirely.

2. We conducted a series of empirical studies and have acquired heuristics used by ontology engineers to resolve inconsistencies in ontologies. The *heuristics* are based on such things as the impact of a removal on the ontology, knowledge of ontology structures, common OWL modelling errors, etc. We find that some heuristics are already incorporated in existing tools; we also discovered new useful heuristics.

3. We propose a *heuristic-based service* to guide users to select axioms which should be removed or modified, in order to help users achieve a satisfiable ontology. The potentially problematic axioms are ranked in order of *confidence*. The confidence of an axiom is calculated from a set of heuristics. The heuristics, obtained from the empirical studies and existing literature, are based on:

   - impact of removal on the ontology,
   - knowledge of ontology structure, and
   - linguistic heuristic

4. We utilise the fine-grained approach to calculate the lost entailments (e.g., subsumption relationships) of named concepts due to the removal of (parts of) axioms. This approach allows us to identify *harmful* and *helpful* changes with respect to a named concept in an ontology. A harmful change cannot resolve the problem, and might cause additional unsatisfiability in the ontology; a helpful change resolves the problem without causing additional contradictions and compensates for the lost entailments.

5. A *caching technique* is proposed to improve the efficiency of consistency checking for modified ontologies. The main idea is to reuse the result of a previous consistency check and

to only compute the consequences derived from a change, avoiding the re-computation of unchanged consequences.

6. We evaluate whether the proposed approaches can improve the process of resolving unsatisfiable ontologies. For this purpose, we incorporated the proposed methods as a plugin of Protégé, called 'RepairTab'. This allows us to analyse in what ways the methods reduce the users' effort in the task of resolving inconsistencies, and in what ways they increase the quality of the modified ontologies with respect to the users' requirements. The following were the main aspects evaluated.

   - In order to determine the effectiveness of our heuristic-based service which guides users to select appropriate axioms for removal or modification, we conducted a usability study with different groups of subjects using different sets of heuristics to resolve inconsistencies in ontologies. Our results show that the heuristics are generally useful for debugging ontologies with common errors and unsatisfiable ontologies merged from two satisfiable ontologies.
   - For the fine-grained approach, we also conducted a usability evaluation to test its effectiveness by comparing with existing ontology debugging tools. The results show that (1) the fine-grained approach greatly facilitated the subjects' understanding of the reasons for concepts' unsatisfiability, and (2) the subjects found the helpful changes very useful to minimise the impact of changes on the ontologies; but the harmful changes were less useful for them.
   - For evaluating the runtime and memory performance of the proposed algorithms, we use a number of existing ontologies to evaluate the efficiency. The results demonstrate that our algorithms provide acceptable performance when used with real world ontologies.

1.5 Outline of the Thesis

The thesis is organised as follows:

- Chapter 2 introduces basic description logic formalisms, reasoning services, reasoning techniques, and OWL ontologies.
- Chapter 3 is the literature review on existing approaches to handling inconsistent ontologies. In particular, we discuss SWOOP's scope and limitations. We also review other approaches related to heuristic-based approaches, and heuristics used in ontology management (including ontology merging, ontology versioning and ontology evolution).
- In Chapter 4 we propose a fine-grained approach to pinpointing problematic parts of axioms associated with an unsatisfiable named concept.
- In Chapter 5 we present an empirical study, in which we acquire a variety of heuristics used by ontology engineers to deal with unsatisfiable ontologies.

- In Chapter 6 we formalise a set of heuristics, derived from the empirical studies and literature work, to evaluate confidence in axioms, so as to suggest to the user which axioms should be removed or modified.

- In Chapter 7, with the fine-grained approach, we are able to calculate the lost entailments due to axiom removal; we also identify changes which are helpful in that they restore lost entailments due to removal of axioms, and those changes that are harmful in that they cause additional unsatisfiability. A caching technique is proposed to improve the efficiency of consistency checking for modified ontologies.

- Chapter 8 presents the implementation details of the approaches presented in Chapters 4–7. We also present the results of usability and performance evaluations which demonstrate the practical significance of our approaches.

- Chapter 9 contains some concluding remarks and describes planned future work.

- In the Appendix, we provide details of the empirical studies conducted.

The main results in Chapters 4 and 7 have previously been published in "Joey SC Lam et al. *A Fine-Grained Approach to Resolving Unsatisfiable Ontologies*. In Proc. of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI-2006), 428 – 434. 2006. (Best Paper Award)".

Chapter 2

Foundations

In this chapter Description Logics (DLs) and Web Ontology Languages (OWL) will be introduced. Section 2.1 describes the syntax and semantics of DLs. Section 2.2 introduces DLs reasoning services. Section 2.3 presents the tableaux-based algorithm with an example. Finally, Section 2.4 describes Web Ontology Language (OWL) with its three species, and the correspondence between OWL and DLs.

2.1 Description Logics (DLs)

Description Logics (DLs) [9] are a family of logic-based knowledge representation formalisms. They represent knowledge of an application domain (i.e., the world) by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals existing in the domain (the world description). DLs support reasoning about the knowledge of an application domain: implicitly represented knowledge can be inferred from the knowledge that is explicitly contained in the knowledge base. For example, if we define that all mothers are female, and Mary is a mother, then the DL system can infer Mary is female.

2.1.1 Syntax and Semantics of Description Languages

The basic "building blocks" used to represent knowledge in DLs are *atomic concepts* (concept names), *atomic roles* (role names) and *individuals*. The sets of concept names, role names and individuals which appear in a knowledge base are denoted as **C**, **R** and **I**, respectively. CN will be used to denote a concept name, RN a role name, and a, b individual names. Concepts are groups of objects which share some common property, such as Mother, Female. Roles represent binary relationships between objects, such as $\mathsf{has\_gender}$, $\mathsf{has\_child}$. Individuals represent the objects themselves, such as Mary, Moon etc. *Concept descriptions* can be built from atomic ones inductively with concept and role constructors. The boolean concept constructors include conjunction ($\sqcap$), disjunction ($\sqcup$) and negation ($\neg$). For example, we can represent a concept 'female person' using a conjunction operator such as $\mathsf{Person} \sqcap \mathsf{Female}$. Besides the booleans, DLs typically provide concept constructors that use roles to form concept descriptions, such as existential restriction constructor ($\exists R.C$), the value restriction constructor ($\forall R.C$), and the number restriction constructor ($\geq n.R$, $\leq n.R$) etc. For example, we can describe a concept 'a person who has married to a doctor and has at least one child' as $\mathsf{Person} \sqcap \exists\ \mathsf{married}.\mathsf{Doctor} \sqcap \geq 1\ \mathsf{has\_child}$.

In this thesis, we mainly focus on concept descriptions built from concepts, intersection, union, complements, universal role quantification, existential role quantifications. This attribute concept description language, is called $\mathcal{ALC}$ [134] ($\mathcal{AL}$ stands for *Attribute language* and $\mathcal{C}$ stands for *Complement*). $\mathcal{ALC}$ is fairly expressive, and has been considered as a basic DL language of interests in numerous studies in DL. $\mathcal{ALC}$ is propositionally complete in that all boolean set operations can be expressed. A $\mathcal{ALC}$ concept (or simply *concept*) is defined by the following syntactic rules, where CN is a concept name, R is a role, C, C_1 and C_2 are concept descriptions, and the concept names $\top$ (top) and $\bot$ (bottom) represent the most general and least general concepts respectively:

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

The meaning of concepts, roles and individuals is given by an interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set of individuals (the *domain* of the interpretation) and an *interpretation function* $(\cdot^{\mathcal{I}})$, which maps each atomic concept $\mathsf{CN} \in \mathbf{C}$ to a set $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each atomic role $R \in \mathbf{R}$ to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function can be extended to give semantics to concept descriptions (see Table 2.1). An interpretation $\mathcal{I}$ is said to be a model of a concept C, or $\mathcal{I}$ models C, if the interpretation of C in $\mathcal{I}$ is not empty.

Constructor	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
concept name	CN	$\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
general negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
exists restriction ($\mathcal{E}$)	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$

Table 2.1: Semantics of $\mathcal{ALC}$-concepts

2.1.2 Knowledge Base

A DL knowledge base Σ typically contains TBox $\mathcal{T}$ and ABox $\mathcal{A}$, formally $\Sigma = \langle \mathcal{T}, \mathcal{A}\rangle$. The TBox introduces the terminological axioms, i.e., which make statements about how concepts and roles are related to each other. The ABox introduces individuals, by giving them names, and asserting properties of these individuals.

TBox

A TBox $\mathcal{T}$ is a finite, possibly empty, set of axioms of the form $C \sqsubseteq D$ or $C \doteq D$. Axioms of the form $C \sqsubseteq D$ are called *concept inclusions* (i.e., *general concept inclusion* axioms, GCIs), while axioms of the form $C \doteq D$ are called *concept equivalence*, which is simply an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$, where C and D are not restricted to be atomic. An axiom is called a defintion of CN if it is of the form $\mathsf{CN} \sqsubseteq D$ or $\mathsf{CN} \doteq D$, where CN is an atomic name, it is *unique* if the KB contains no other definition of CN.

Definitions of the form $\mathsf{CN} \sqsubseteq D$ are sometimes called *primitive* or necessary, as D specifies a necessary condition for individuals of CN, while those of the form $\mathsf{CN} \doteq D$ are sometimes called *non-primitive* or necessary and sufficient as D specifies conditions that are both necessary and sufficient for individuals of CN.

Example 11 (A Non-primitive Concept) *The axiom:* $\mathsf{Woman} \doteq \mathsf{Person} \sqcap \mathsf{Female}$ *introduces the definition of* Woman *and states that a* Woman *is necessarily both a* Person *and* Female*, and if an instance is both a* Person *and* Female *this is sufficient to infer that it is a* Woman*. That is,* $\mathsf{Person} \sqcap \mathsf{Female}$ *is said to be the definition of* Woman.

Example 12 (A GCI) *We can assert only professors but not students can teach courses as* $\exists\mathsf{teaches.Course} \sqsubseteq \neg\,\mathsf{Student} \sqcup \mathsf{Professor}$*, which is a general concept inclusion axiom.*

A TBox is called unfoldable if the left-hand side of every $\alpha \in \mathcal{T}$ contains a concept name CN, there are no other αs with CN on the left-hand side, and the right-hand side of α contains no direct or indirect references to CN (no cycles) [114]. We divide $\mathcal{T}$ into an unfoldable part $\mathcal{T}_u$ and a general part $\mathcal{T}_g$, such that $\mathcal{T}_g = \mathcal{T} \setminus \mathcal{T}_u$.

An interpretation $\mathcal{I}$ *satisfies* a concept inclusion $C \sqsubseteq D$, or $\mathcal{I}$ models $C \sqsubseteq D$ (written as $\mathcal{I} \vDash C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ satisfies a concept equivalence $C \doteq D$ (written as $\mathcal{I} \vDash C \doteq D$) iff $C^{\mathcal{I}} = D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model for a TBox $\mathcal{T}$ (written as $\mathcal{I} \vDash \mathcal{T}$) iff $\mathcal{I}$ satisfies all the axioms in $\mathcal{T}$.

ABox

The second component of a knowledge base, in addition to the terminology or TBox, is the world description or ABox. The ABox contains extensional knowledge about the domain of interest, that is, assertions about individuals. In the ABox, one introduces individuals, by giving them names, and one asserts properties of these individuals. Given individual names a, b, c, using concepts C and roles R, one can make assertions of the following two kinds in an ABox: $a : C$ and $R(b, c)$. The first kind, called *concept assertions*, states that a belongs to (the interpretation of) C; the second axiom, called *role assertions*, states that c is a filler of the relation R for b.

Example 13 *The statement* Anna is a female person *can be asserted as* $\mathsf{Anna} : (\mathsf{Female} \sqcap \mathsf{Person})$*, that is, a concept assertion. The statement* Anna has Peter as a child *can be asserted as* $\mathsf{hasChild}(\mathsf{Anna}, \mathsf{Peter})$*, that is, a role assertion.*

An interpretation $\mathcal{I}$ satisfies a concept assertion $a : C$ (written as $\mathcal{I} \vDash a : C$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it satisfies a role assertion $R(a, b)$ (written as $\mathcal{I} \vDash R(a, b)$) iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is said to be a model of the ABox $\mathcal{A}$ iff every assertion of $\mathcal{A}$ is satisfied by $\mathcal{I}$. The ABox $\mathcal{A}$ is said to be *satisfiable* iff it admits a model.

2.2 Reasoning

A DL system not only stores terminologies and assertions, but also offers services that *reason* about them. Reasoning with a DL knowledge base is a process of discovering implicit knowledge entailed by the knowledge base. Typically, reasoning services for a TBox are to determine whether a concept description is *satisfiable* (i.e., non-contradictory), or to determine if one concept description is more general than another one (i.e., whether the first *subsumes* the second). Reasoning services for an ABox are to check whether its set of assertions is *consistent* (i.e., whether it has a model), or to check whether an individual is an *instance* of a given concept description. We summarise the reasoning services for DLs as follows:

- **Consistency Checking**, which ensures that an ontology does not contain any contradictory facts. A knowledge base Σ is consistent iff there exists a model $\mathcal{I}$ of Σ.

- **Concept Satisfiability**, which checks if it is possible for a concept to have any individuals. A concept C is satisfiable with respect to $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is non-empty. We say that $\mathcal{I}$ is a model of C.

- **Subsumption**, which computes the subsumption relations between named concepts. A concept C is subsumed by a concept D with respect to $\mathcal{T}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. We write $\mathcal{T} \models C \sqsubseteq D$.

- **Instantiation**, which finds the most specific concept that an individual belongs to. An individual a is an instance of a concept C in a knowledge base Σ, iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ in every model $\mathcal{I}$ of Σ, i.e., $\Sigma \models a : C$.

Example 14 *Consider the knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, $\mathcal{T}$ consists of the following axioms:*
1. $\mathsf{Parent} \doteq \mathsf{Person} \sqcap \exists \mathsf{hasChild}.\mathsf{Person}$
2. $\mathsf{Mother} \doteq \mathsf{Parent} \sqcap \mathsf{Female}$

$\mathcal{A}$ consists of the following axioms:
3. Mary*:*Mother
4. David*:*Person
5. Peter*:*Person
6. hasChild*(*Peter,David*)*

Let $\mathcal{I}$ be an interpretation with universe $\Delta^{\mathcal{I}} = \{a, b, c, d\}$ and an interpretation function that interprets the primitive concepts, roles and individuals from $\mathcal{T}$ and $\mathcal{A}$ as follows:
$\mathsf{Person}^{\mathcal{I}} = \{a, b, c, d\}$, $\mathsf{Female}^{\mathcal{I}} = \{a, d\}$, $\mathsf{Parent}^{\mathcal{I}} = \{a, b\}$, $\mathsf{Mother}^{\mathcal{I}} = \{a\}$, $\mathsf{Mary}^{\mathcal{I}} = a$, $\mathsf{Peter}^{\mathcal{I}} = b$, $\mathsf{David}^{\mathcal{I}} = c$, $\mathsf{hasChild}^{\mathcal{I}} = \{(a, d), (b, c)\}$,
then $\mathcal{I}$ is a model of $\mathcal{T}$ together with $\mathcal{A}$. Note that in $\mathcal{I}$, Peter *is a parent although the ABox does not explicitly specify this.*

In this thesis, we will mostly focus on concept satisfiability and subsumption reasoning because they have been more widely used in DL applications. Note that in languages that are propositionally closed (i.e., that provide, either implicitly or explicitly, conjunction, union and negation of class descriptions), such as OWL-DL and OWL-Lite, the problems of ontology consistency, concept satisfiability, concept subsumption and instance checking can be reduced to each other [14]:

- Subsumption reduced to unsatisfiability – $\mathcal{T} \models C \sqsubseteq D$, iff, in all interpretations $\mathcal{I}$ satisfies $\mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and so $C^{\mathcal{I}} \cap (\neg D)^{\mathcal{I}} = \emptyset$, that is,

$$\mathcal{T} \models C \sqsubseteq D \text{ iff } \mathcal{T} \models \neg(C \sqcap \neg D)$$

- Unsatisfiability reduced to subsumption – C is not satisfiable with respect to $\mathcal{T}$, iff, in all $\mathcal{I}$ that satisfy $\mathcal{T}$, $C^{\mathcal{I}} = \emptyset$ and so $C^{\mathcal{I}} \subseteq \bot^{\mathcal{I}}$, that is,

$$\mathcal{T} \models \neg C \text{ iff } \mathcal{T} \models C \sqsubseteq \bot$$

- Concept satisfiability reduced to ABox consistency – C is satisfiable iff the ABox $\{(a : C)\}$ for some $a \in$**I** is consistent.

- Instantiation reduced to consistency checking – a is an instance of C w.r.t. $\mathcal{A}$ iff $\mathcal{A} \cup \{\neg(a : C)\}$ is inconsistent.

We describe the algorithm for checking concept satisfiability and subsumption in the next section.

2.3 A Tableau Algorithm for $\mathcal{ALC}$

The first tableau-based algorithm was presented by Schmidt-Schauß and Smolka [134] for satisfiability of $\mathcal{ALC}$-concepts. This approach has been extended for more expressive DL languages (see e.g., [80; 6; 79]). In this section, we will present a tableau-based satisfiability algorithm for $\mathcal{ALC}$-concepts [74]; an example illustrates how it works.

A tableau-based algorithm decides the satisfiability of a concept description w.r.t. an ontology by trying to construct a representation of a model for it, called a tree **T**. Each node x in the tree is labelled with a set $\mathcal{L}(x)$ of concept assertions $(a : C)$ or role assertions $R(a, b)$. Each concept assertion must satisfy, i.e.,

$$\text{if } a : C \in \mathcal{L}(x), \text{ then } a \in C^{\mathcal{I}}$$

A role assertion represents the individual a has an R relationship with b, it must satisfy:

$$\text{if } R(a, b) \in \mathcal{L}(x), \text{ then } \langle a, b\rangle \in R^{\mathcal{I}}$$

Example 15 *Let there be an ontology containing the single axiom $A \sqsubseteq \exists R.B$. The meaning of the axiom is that all instances of A participate in an R relationship with at least one of the instances of B. We now build a tree to describe the semantics. A root node x is labelled with $\mathcal{L}(x) = \{a : A\}$, which means a is an individual of A. To represent the definition of A in the tree, we add a new element $a : \exists R.B$ to $\mathcal{L}(x)$. Next, to represent all instances of A participating in an R relationship with at least one of the instances of B, we create a new individual b which belongs to B, and add the new elements $R(a, b)$ and $b : B$ to the label of the node (see Figure 2.1).*

$$\textcircled{x}\ \mathcal{L}(x) := \{a : A, a : \exists R.B, R(a, b), b : B\}$$

Figure 2.1: A tree with a root node x.

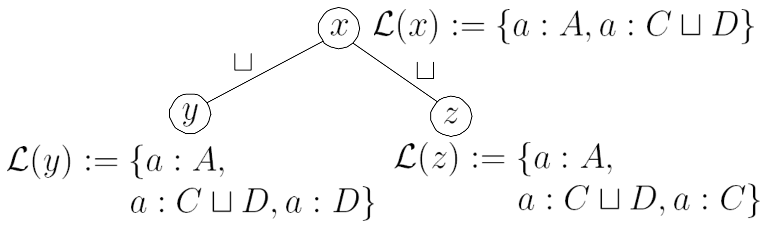

$\mathcal{L}(x) := \{a : A, a : C \sqcup D\}$
$\sqcup$
$\sqcup$
$\mathcal{L}(y) := \{a : A, a : C \sqcup D, a : D\}$
$\mathcal{L}(z) := \{a : A, a : C \sqcup D, a : C\}$

Figure 2.2: A tree with two leaf nodes y and z

Example 16 *Given an ontology containing the single axiom $A \sqsubseteq C \sqcup D$. The meaning of the axiom is that all instances of A belong to the union set of the instances of C and D. We now build a tree to describe its semantics. A root node x is labelled with $\mathcal{L}(x) = \{a : A\}$, which means a is an individual of A. To represent the definition of A in the tree, we add a new element $a : (C \sqcup D)$ to $\mathcal{L}(x)$. Then, we add two $\sqcup$-successor nodes, y and z, which represent instances of A which either belong to C or D. Therefore nodes y and z are labelled with $\{a : A, a : C \sqcup D, a : C\}$ and $\{a : A, a : C \sqcup D, a : D\}$ respectively (see Figure 2.2).*

2.3.1 Applications of Expansion Rules

To determine the satisfiability of a concept C in an ontology, a tree **T** is initialised to contain a single node x, with $\mathcal{L}(x) = \{(a : C)\}$. The tree is then built by applying a set of expansion rules that either extend node labels or add new leaf nodes. The set of expansion rules for the $\mathcal{ALC}$ Description Logic is shown in Table 2.2. where A_i is a concept name, C, C_1, C_2 are concept descriptions, R is a role name, a and b are individuals.

$U^+_{\doteq}$-rule	if $A_i \doteq C_i \in \mathcal{T}_u$, $a : A_i \in \mathcal{L}(x)$ and $a : C_i \notin \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{a : C_i\}$
$U^-_{\doteq}$-rule	if $A_i \doteq C_i \in \mathcal{T}_u$, $a : \neg A_i \in \mathcal{L}(x)$ and $a : \neg C_i \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{a : \neg C_i)\}$
$U_{\sqsubseteq}$-rule	if $A_i \sqsubseteq C_i \in \mathcal{T}_u$, $a : A_i \in \mathcal{L}(x)$ and $a : C_i \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{a : C_i\}$
$\sqcap$-rule	if $(a : C_1 \sqcap C_2) \in \mathcal{L}(x)$, and $\{a : C_1, a : C_2\} \subsetneq \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cap \{a : C_1, a : C_2\}$
$\sqcup$-rule	if $(a : C_1 \sqcup C_2) \in \mathcal{L}(x)$, and $\{a : C_1, a : C_2\} \cap \mathcal{L}(x) = \emptyset$ then create two $\sqcup$-successors y, z of x with $\mathcal{L}(y) := \mathcal{L}(x) \cup \{a : C_1\}$, $\mathcal{L}(z) := \mathcal{L}(x) \cap \{a : C_2\}$
$\exists$-rule	if $a : \exists R.C \in \mathcal{L}(x)$, a is not blocked, and there is no b s.t. $\{R(a,b), b : C\} \nsubseteq \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{R(a,b), b : C\}$, where b is an individual name not occurring in $\mathcal{L}(x)$
$\forall$-rule	if $a : \forall R.C \in \mathcal{L}(x)$ and there is some b s.t. $R(a,b) \in \mathcal{L}(x)$ and $b : C \notin \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{b : C\}$
$\sqsubseteq$-rule	if $(C_i \sqsubseteq D_i) \in \mathcal{T}_g$, a is not blocked, and $a : \neg C_i \sqcup D_i \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{a : \neg C_i \sqcup D_i\}$, for every individual a in $\mathcal{L}(x)$

Table 2.2: Tableaux Expansion Rules for $\mathcal{ALC}$

We now explain the expansion rules. The three rules ($U^+_{\doteq}$-rule, $U^-_{\doteq}$-rule and $U_{\sqsubseteq}$-rule) describe the *unfolding* procedure. Unfolding a concept description is to replace defined names by their definitions, so that it does not contain names defined in the terminology. These rules are used for optimisation (also called lazy unfolding) [12]. That means only unfolding concepts as required by the progress of the satisfiability testing algorithm. The $U^+_{\doteq}$-rule and $U^-_{\doteq}$-rule reflect the symmetry of the equality relation in the non-primitive definition $A \doteq C$, which is equivalent to $A \sqsubseteq C$ and $\neg A \sqsubseteq \neg C$. The $U_{\sqsubseteq}$-rule on the other hand reflects the asymmetry of the subsumption relation in the primitive definition $A \sqsubseteq C$.

Disjunction concept elements $(a : C_1 \sqcup C_2) \in \mathcal{L}(x)$ result in *non-deterministic* expansion. We deal with this non-determinism by creating two $\sqcup$-successors y, z of x with: $\mathcal{L}(y) := \mathcal{L}(x) \cup \{a : C_1\}$, and $\mathcal{L}(z) := \mathcal{L}(x) \cup \{a : C_2\}$. For a conjunction concept element $(a : C_1 \sqcap C_2) \in \mathcal{L}(x)$, it means b is an instance of both C_1 and C_2, therefore, $(a : C_1)$ and $(a : C_2)$ are both added to $\mathcal{L}(x)$.

Each existential role restriction concept $(a : \exists R.C) \in \mathcal{L}(x)$ causes the introduction of new elements $R(a, b)$ and $b : C$, where b is an individual name not occurring in $\mathcal{L}(x)$. A universal role restriction concept $a : \forall R.D$ extends the label with a new concept element $b : D$ if $R(a, b)$ exists in $\mathcal{L}(x)$.

The $\sqsubseteq$-rule handles the consistency checking of an ontology with GCIs. A GCI $C \sqsubseteq D \in \mathcal{T}$ is satisfied by a model $\mathcal{I}$ iff every individual in the model satisfies $\neg C \sqcup D$ [24]. That means

$$C \sqsubseteq D \text{ is satisfied by } \mathcal{I} \text{ iff } (\neg C \sqcup D)^{\mathcal{I}} = \Delta^{\mathcal{I}}$$

The tree constructed by the tableau algorithm is therefore restricted to those interpretations $\mathcal{I}$ which represent models satisfying the GCI $C \sqsubseteq D$ by imposing the following constraint:

$$\text{for all individuals } a \text{ in } \mathcal{L}(x),\ (a : \neg C \sqcup D) \in \mathcal{L}(x)$$

During the expansion, concept descriptions are assumed to be converted to negation normal form. A concept description is in negation normal form when negations apply only to concept names, and not to compound terms. This can be done by using de Morgan's rules and usual rules for quantifiers. For example, $\neg\forall R.C = \exists R.\neg C$, $\neg\exists R.C = \forall R.\neg C$. We repeatedly expand the tree by applying the rules in Table 2.2 as many times as possible. A node is fully expanded when none of the rules can be applied to it. A tree **T** is fully expanded when all of its leaf nodes are fully expanded. **T** contains an obvious *clash* when, for some individual a and some concept C, $\{a : C, a : \neg C\} \subseteq \mathcal{L}(x)$. When a clash is found in a node, the tableau algorithm [11] will either backtrack and select a different leaf node, or report the clash and terminate, if no node remains to be expanded. If the input of the tableau algorithm is a concept C in a terminology $\mathcal{T}$, we have the following property: C is *unsatisfiable* iff all paths from the root to a leaf node in the tree contain at least one clash.

Let us use an example to illustrate the tableaux-based algorithm.

Example 17 *Let us assume that an ontology $\mathcal{O}$ contains the following axioms:*
α_1*:* $A \doteq C \sqcap \forall R.B$
α_2*:* $C \doteq \exists R.\neg B \sqcap B$
α_3*:* $G \doteq \forall R.(C \sqcap F)$

Figure 4.2 shows how the tableau algorithm is applied on Example 17 to check for the satisfiability of A. The tree **T** contains a root node x which has a clash because $\{b : \neg B, b : B\} \subseteq \mathcal{L}(x)$.

(1) Initialise the root node x with $\mathcal{L}(x) := \{a : A\}$,

(2) Apply the $U^{+}_{\doteq}$-rule to $a : A \in \mathcal{L}(x)$, yielding
$\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C \sqcap \forall R.B\}$,

(3) Apply the $\sqcap$-rule to $a : C \sqcap \forall R.B \in \mathcal{L}(a)$, yielding
$\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C, a : \forall R.B\}$,

(4) Apply the $U^{+}_{\doteq}$-rule to $a : C \in \mathcal{L}(a)$, followed by applying the $\sqcap$-rule, yielding
$\mathcal{L}(x) := \mathcal{L}(x) \cup \{a : \exists R.\neg B \sqcap B, a : \exists R.\neg B, a : B\}$,

(5) Apply the $\exists$-rule to $a : \exists R.\neg B \in \mathcal{L}(x)$, followed by applying the $\forall$-rule, obtaining
$\mathcal{L}(x) := \mathcal{L}(x) \cup \{b : \neg B, b : B\}$,
where b is an individual name not occurring in $\mathcal{L}(x)$;

(6) The algorithm terminates as a clash is detected in the the root node x, $\{b : \neg B, b : B\} \subseteq \mathcal{L}(x)$

Figure 2.3: The application of tableaux expansion rules on A in Example 17

2.3.2 Blocking

To deal with cyclic axioms, it is necessary to add cycle detection (often called *blocking*) to the preconditions of some of the expansion rules in order to guarantee termination [8; 24]. We use a simple example to describe the need for blocking:

Example 18 *Given a $\mathcal{T}$ containing a single cyclic axiom, α_1: $A \doteq \exists R.A$, then testing the satisfiability of A leads to:*
Step 1. $\mathcal{L}(x) := \{a_0 : A, a_0 : \exists R.A\}$
Step 2. $\mathcal{L}(x) := \mathcal{L}(x) \cup \{R(a_0, a_1), a_1 : A, a_1 : \exists R.A\}$
Step 3. $\mathcal{L}(x) := \mathcal{L}(x) \cup \{R(a_1, a_2), a_2 : A, a_2 : \exists R.A\}$
$\cdots$

$\mathcal{L}(x)$ is initialised to contain $a_0 : A$. The application of the $U^{+}_{\doteq}$-rule leads to $a_0 : \exists R.A$ being added to $\mathcal{L}(x)$ (Step 1). The application of the $\exists$-rule leads to the creation of a new individual a_1 at Step 2, new elements $R(a_0, a_1)$ and $a_1 : A$ are added to the label of the node. The same expansion rules would be applied and the process would continue indefinitely. Since all individuals a_1, a_2, $\cdots$ receive the same concept assertions as a_0, we may say the algorithm has run into a cycle. Therefore, blocking is necessary to ensure termination. The general idea is to stop the expansion of a node whenever the same concept assertions recur in the node. Blocking imposes a condition on the $\exists$-rule: an individual a is *blocked* by an individual b in a node label $\mathcal{L}(x)$ iff $\{D|(a : D) \in \mathcal{L}(x)\} \subseteq \{D'|(b : D') \in \mathcal{L}(x)\}$. In our example, that would mean individual a_1 in $\mathcal{L}(x)$ at Step 2 is blocked by a_0, because $\{A, \exists R.A\} \subseteq \{A, \exists R.A\}$ in the label of the node. Intuitively, it can be seen that termination is now guaranteed because a finite terminology can only produce a finite number of different concept descriptions and therefore a finite number of different labelling sets; all nodes must therefore eventually be blocked.

2.3.3 Optimisations

Subsumption testing in DLs is known to be intractable in the worst case due to non-determinism [74]. In order to provide acceptable performance with realistic large knowledge bases, existing DL reasoners, such as FaCT++ [146], RACER [60] and Pellet [136], implement a variety of optimisation techniques [135; 9; 79; 61; 59]. These optimisation techniques make a dramatic difference to the performance of a description logic system. We briefly describe some of the optimisations, which are mentioned in the rest of the thesis.

1. *Normalisation and Simplification*: Tableaux algorithms generally assume that the concept expression to be tested is in negation normal form, with negations applying only to atomic concepts [72; 10; 24]. Normalisation helps to identify obvious contradictions, by syntactically comparing descriptions in their normal form. For example, $\neg(C_1 \sqcup C_2) \sqcap C_2$ is normalised as $\neg C_1 \sqcap \neg C_2 \sqcap C_2$, a clash is then detected immediately without expanding the disjunction concept. Normalisation can also include a range of simplifications so that syntactically obvious contradictions are detected. For example, $\exists R.\bot$ can be simplified to $\bot$; $\forall R.\top$ can be simplified to $\top$.

2. *Lazy Unfolding*: Unfolding a concept description is to replace defined concept names by their definitions, so that concept descriptions do not contain concept names defined in the terminology. Lazy unfolding means only unfolding concepts as required by the progress of the subsumption or satisfiability testing algorithm [12]. With a tableau algorithm, a defined concept is only unfolded when it occurs in a node label. With lazy unfolding, the algorithm avoids unnecessary unfolding of irrelevant subconcepts, either because a contradiction is discovered without fully expanding the tree, or because a non-deterministic expansion leads to a complete and clash-free tree.

3. *Absorption*: GCIs are a major cause of intractability: each GCI $C \sqsubseteq D$ causes a disjunction concept $\neg C \sqcup D$ to be added to every individual in the nodes of the tree. This leads to an exponential increase in the size of the search space to be explored by the $\sqcup$-rule [74]. Absorption is a technique that tries to eliminate GCIs as possible from a KB by replacing them with primitive definitions. Given a general inclusion axiom $C \sqsubseteq D$ (where C is any concept description), the idea is to manipulate the axiom to have the form of a primitive definition $A \sqsubseteq D'$ (where A is an atomic concept), using the equivalences in Figure 2.4. This axiom can be merged into an existing primitive definition $A \sqsubseteq C'$ to give $A \sqsubseteq C' \sqcap D'$. The advantage of absorption is that it avoids adding large number of irrelevant disjunctions to every individual in the nodes of the tree constructed by the tableau algorithm.

$$\mathsf{CN} \sqsubseteq C \text{ and } \mathsf{CN} \sqsubseteq D \text{ iff } \mathsf{CN} \sqsubseteq C \sqcap D$$
$$\mathsf{CN} \sqcap C \sqsubseteq D \text{ iff } \mathsf{CN} \sqsubseteq D \sqcup \neg C$$
$$C \sqsubseteq \neg\mathsf{CN} \sqcup D \text{ iff } \mathsf{CN} \sqsubseteq D \sqcup \neg C$$

Figure 2.4: Axiom equivalences used in absorption

Example 19 *Given an ontology consisting of the following axioms:*
1. $A \doteq \forall R.\neg C \sqcap D$
2. $D \sqsubseteq \neg E$
3. $\forall R.\neg C \sqsubseteq \neg D \sqcup \neg E$

Axiom 3 can be absorbed as $D \sqsubseteq \neg E \sqcup \exists R.C$*, then axioms 2 and 3 can be merged as* $D \sqsubseteq (\neg E \sqcup \exists R.C) \sqcap \neg E$*. Therefore, the concept* $(\neg E \sqcup \exists R.C) \sqcap \neg E$ *will be added to the node of the tree automatically when* D *is unfolded.*

2.4 Web Ontology Language (OWL)

High quality ontologies are crucial for the Semantic Web; high quality ontologies depend on the availability of formalisms with well-defined semantics and powerful reasoning tools. Description Logics provide the Semantic Web with a common understanding of the basic semantic concepts used to annotate the Web resources; they also provide their users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge. W3C (the World Wide Web Consortium) standardised the OWL Web Ontology Language in its Semantic Web Activity in 2004 [18], whereas OWL DL is based on $\mathcal{SHOIN}(\mathbf{D})$ [76]. OWL DL benefits from many years of DL research, which include well defined semantics, known reasoning algorithms, highly optimised systems, and well understood formal properties (such as complexity and decidability) [13].

OWL has three increasingly expressive sub-languages: OWL Lite, OWL DL and OWL Full. OWL Lite is the syntactically simplest sub-language. It supports those users primarily needing a classification hierarchy and simple constraint features. For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1. The advantage of this is a language that provides a minimal useful subset of language features, which are relatively straightforward for tool developers to implement. The disadvantaged is of course a restricted expressivity. OWL Lite corresponds to an expressive description logic $\mathcal{SHIF}(\mathbf{D})$. OWL DL is more expressive than OWL Lite and corresponds to the description logic $\mathcal{SHOIN}(\mathbf{D})$. It is possible to automatically compute class hierarchy and consistency checking in an ontology that conforms to OWL DL. OWL Lite and OWL DL are both decidable, and the complexity of the ontology entailment problems of OWL Lite and OWL DL is EXPTIME-complete and NEXPTIME-complete, respectively [77]. OWL Full is the most expressive OWL sub-language with no computational guarantees. In OWL Full, all RDF and RDFS combinations are allowed. For example, a class can be treated simultaneously as a collection of individuals and as an individual in its own right. From a modelling and semantic point of view, OWL shares a strong correspondence with Description Logics borrowing many logical constructs as shown in Table 2.3 [78].

OWL Constructor	**DL Syntax**	**Example**
owl:equivalentTo(C_1, C_2)	$C_1 \doteq C_2$	Father $\doteq$ Male_Parent
rdfs:subClassOf(C_1, C_2)	$C_1 \sqsubseteq C_2$	Female $\sqsubseteq$ Person
owl:disjointWith(C_1, C_2)	$C_1 \sqsubseteq \neg C_2$	Female $\sqsubseteq \neg$ Male
owl:intersectionOf(C_1, C_2)	$C_1 \sqcap C_2$	Human $\sqcap$ Parent
owl:unionOf(C_1, C_2)	$C_1 \sqcup C_2$	Female $\sqcup$ Parent
owl:complementOf(C)	$\neg C$	$\neg$ Female
owl:oneOf(o_1, o_2)	$\{o_1\} \cup \{o_2\}$	{Mary} $\cup$ {Peter}
owl:someValuesFrom(R, C)	$\exists R.C$	$\exists$ has_child.Person
owl:allValuesFrom(R, C)	$\forall R.C$	$\forall$ has_daugther.Female
owl:hasValue(R, o)	$\exists R.\{o\}$	$\exists$ has_child.{Peter}
owl:maxCardinality(R, n)	$\leq n.R$	$\leq$ 3.has_child
owl:minCardinality(R, n)	$\geq n.R$	$\geq$ 4.has_child
owl:cardinality(R, n)	$= n.R$	= 2.has_child

Table 2.3: Correspondence between OWL and DL

In contrast to the usual DL syntax sketched previously, OWL ontologies are represented in RDF-syntax derived from XML. The following example shows a concept definition in OWL.

Example 20 *Given an axiom* Woman $\doteq$ Person $\sqcap$ Female, *the corresponding definition in an OWL ontology in standard syntax would be as follows.*

```
⟨owl:Class rdf:ID="Woman"⟩
   ⟨owl:intersectionOf rdf:parseType="Collection"⟩
      ⟨owl:Class rdf:about="#Person"/⟩
      ⟨owl:Class rdf:about="#Female"/⟩
   ⟨/owl:intersectionOf⟩
⟨/owl:Class⟩
```

As described in the OWL Web Ontology Language Guide[1], OWL concepts are called 'classes', roles are called 'properties', and conjunction is called 'intersection'. An existential restriction is denoted by a 'restriction' of the type 'hasValue' – as opposed to 'allValuesFrom' for value restrictions.

2.4.1 Three Properties in OWL DL

The following properties of OWL ontologies are the principal hurdles facing novice ontology users when they try to understand OWL and Description Logics [125]:

- Reasoning in OWL (description logics) is based on the *open world assumption* (OWA). The open world assumption assumes that its knowledge of the world is incomplete. This implies that something is false only if it can be proved to contradict other information in the ontology. Consider this example[2]: Patricia has a friend who is Arthur. Arthur is a student and he is happy. That is, hasFriend(Patricia, Arthur), Arthur : Student, Arthur : Happy.

[1] http://www.w3.org/TR/owl-guide/
[2] The example is taken from http://owl.man.ac.uk/2005/07/sssw/

However, we cannot infer that Patricia is an instance of $\forall$hasFriend.(Student $\sqcap$ Happy). This is because we cannot assume that if we do not know something then it is false. In this example, there may be other friends that Patricia has that are not students.

- OWL does not have a *unique name assumption* (UNA) whereas in real life most people expect named things to be distinct. If two names are different, it does not mean that they refer to different individuals; we have to explicitly make `owl:differentFrom` or `owl:sameAs` assertions to declare that individuals are different or the same. For example, an ontology contains the assertions: has_capital(China, Beijing), has_capital(China, Shanghai), has_capital is a functional role (i.e., the role has no more than one value for each individual). It means that one country can only have one capital, but the user asserts that both Beijing and Shanghai are the capital of China. However, the reasoner will not detect this ontology to be inconsistent, as Beijing and Shanghai might be the same objects. We have to assert that `owl:differentFrom`(Beijing, Shanghai) to explicitly state that Beijing and Shanghai are different objects, and then the system detects an inconsistency.

- Most users without a formal background in logic, but with a strong background in databases and object-oriented systems intuitively expect domains and ranges to specify the *constraints* that must be fulfilled while populating ontology instances [142]. Suppose that property_R has the concept_C as domain, then if instance_A is not known to be an instance of concept_C, by the closed world assumption it is assumed that instance_A is not an instance of concept_C; hence property_R cannot be instantiated for instance_A in the first place. However, in the OWL languages, domain and range specifications are *axioms* specifying sufficient conditions for an instance to be a member of some class. They are used in reasoning, and may cause a class to be unsatisfiable, or cause a class to be subsumed by another class unexpectedly. That means, if instance_A is not explicitly stated to be an instance of concept_C, but if it has property_R instantiated and property_R has concept_C as domain, then it can be inferred that instance_A is an instance of concept_C.

Example 21 *For the koala.owl example*[3], Koala *is unsatisfiable due to the following axioms:*
1. Koala $\sqsubseteq$ Marsupials
2. Koala $\sqsubseteq$ $\exists$ isHardWorking.*"false"*$\langle$ *xsd:boolean*$\rangle$
3. domain(isHardWorking) = Person
4. Person $\sqsubseteq$ $\neg$Marsupials

The domain of isHardWorking *is* Person*; the ontology modeller asserts that* Koala *has the role* isHardWorking *whose value is false. Hence,* Koala *will be inferred to be a subclass of* Person*. However,* Marsupials*, the superclass of* Koala*, is disjoint with* Person*, and this makes* Koala *unsatisfiable.*

[3]http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

Chapter 3

Literature Review

As mentioned in Chapter 1, ontologies play an important role in Semantic Web applications; it is essential to ensure that they are well-designed, structured in a principled way, and generally of high quality. In the literature, there exist several methods and techniques that allow us to guarantee a high quality of ontologies, in which consistency of ontologies is one of the crucial factors. We firstly review the notions of consistency in ontologies in Section 3.1, and then describe the existing approaches to handling inconsistencies in ontologies in Section 3.2. The techniques which diagnose and debug such inconsistencies are discussed and compared in Section 3.3.

Having reviewed the existing work on debugging ontologies, we wish to provide further support for users to resolve such inconsistencies in an ontology debugging tool. We notice that, SWOOP, a stand-alone ontology editor, already provides users with debugging and repair services. Therefore, in Section 3.4, we give a thorough review of SWOOP's scope and limitations in order to distinguish between this closely related work and our research direction.

As we are also interested in acquiring humans' heuristics to resolve inconsistencies in ontologies, we describe previous work about knowledge elicitation from experts in Section 3.5.1, and then review empirical studies in ontology engineering in Section 3.5.2. Heuristics used in ontology management are presented in Section 3.5.3.

Finally, the summary of the chapter and our research direction are presented in Section 3.6.

3.1 Notions of Consistency

In this section, we describe three notions of ontology consistency distinguished by [63]: structural consistency, logical consistency and user-defined consistency:

- *Structural Consistency*: it ensures that the ontology obeys the constraints of the ontology language with respect to how the constructs of the ontology language are used. For example, OWL Lite disallows the use of owl:oneOf, owl:unionOf, owl:complementOf and owl:disjointWith etc. RDF validators[1] provided by the W3C checks RDF/XML content for syntactic errors. It indicates the lines in the RDF file causing the file to be syntactically invalid. OWL Ontology Validator [2] identifies an ontology as OWL Lite, OWL DL, or OWL Full.

[1]http://www.w3.org/RDF/Validator
[2]http://phoebus.cs.man.ac.uk:9999/OWL/Validator

- *Logical Consistency*: OWL ontologies are formulated in logic-based languages (DLs). An OWL ontology is considered as logically consistent if it does not contain contradictory information. For example, there is no individual belonging to a class and its complement in the ontology; a concept is considered satisfiable if it can have at least one individual in the ontology.

- *User-defined Consistency*: this form of consistency allows users to add their own conditions, the conditions must be met in order for the ontology to be considered consistent. For example, the user can apply the OntoClean methodology in order to detect formal errors in a given taxonomy. OntoClean [155; 57] formalises a set of meta-properties representing the philosophical notions of *rigidity*, *identity*, *unity* and *dependence*. These meta-properties are assigned to properties (i.e., concepts in DL terminology) of the ontology. The meta-properties impose constraints on the subsumption relation, which can be used to check the ontological consistency of taxonomic links. A property is *rigid* ($+R$) if it is essential to all its possible instances; an instance of a rigid property cannot stop being an instance of that property in a different world. For example, the property *being a human* is typically rigid, every human is necessarily so. There are also non-rigid ($-R$) properties, which can acquire or lose (some of) their instances depending on the state of affairs at hand. For example, *Red* is judged to be $-R$ since some instances of it may be necessarily so, and most will not. A further distinction is made among non-rigid properties, namely the introduction of anti-rigid ($\sim R$) properties, which are not essential to all their instances. For example, the property *being a student* is typically anti-rigid, as every instance of student can cease to be such in a suitable state of affairs. One of these constraints is that anti-rigid properties cannot subsume rigid properties. For example, being a person is usually conceptualized as rigid; being a student is typically anti-rigid, as every instance of student may also become a non-student. If the class student subsumes the class person, then contradiction occurs. This is because every instance of student may stop being a student, while no instance of person can stop being a person, this contradicts the previous sentence.

In this thesis, we only focus on the problem of logical consistency, that is, we are interested in ontologies containing contradictory information. For example, if a class has two superclasses which are disjoint with each other in an ontology, then the class is unsatisfiable.

3.2 Approaches to Handling Inconsistency in Ontologies

In the literature, four main approaches to handling inconsistency in ontologies are identified [65]. The first one is *consistent ontology evolution* which guarantees the consistency of ontologies with the presence of changes; the second one is *multi-version reasoning* which deals with inconsistencies by managing different versions of ontologies; the third one is *reasoning with inconsistent ontologies* which avoids the inconsistency and applies a non-standard reasoning method to obtain meaningful answers [84]. The final approach is to *diagnose and repair ontologies* when an inconsistency is detected [131; 154; 93]. The first three approaches will be discussed briefly below, while the fourth approach will be covered in Section 3.3, as it is most relevant to our work.

3.2.1 Consistent Ontology Evolution

Consistent ontology evolution ensures the consistency of ontologies in the presence of changes with respect to a given notion of consistency [63]. The consistency of an ontology is defined in terms of *invariants* that need to be satisfied by the ontology. Stojanovic [142] introduces the *semantics of change* phase in the ontology evolution process. Their aims are (1) to prevent illegal changes i.e., changes that would cause inconsistencies; and (2) to ensure the preservation of consistency in the case that the request can be applied. The task of this phase is to enable the resolution of changes in a systematic manner by ensuring the consistency of the whole ontology. This can be done by completing required changes with additional changes, which guarantee the transfer of the initial consistent ontology into another consistent state. For example, deleting a concept C will cause its subconcepts, some properties and instances to be inconsistent since the concept is referenced somewhere, while it does not exist anymore. The suggested additional changes could (i) delete the orphaned subconcepts of C; (ii) connect the orphaned subconcepts of C the superclasses of C, or (iii) connect the orphaned subconcepts of C to the root concept of the hierarchy.

Flouris et al. [44; 46; 40; 41] point out the weakness of the current ontology evolution approaches. Current work on ontology evolution aims to help users perform changes in ontologies manually rather than performing the changes automatically. They explained it is unrealistic to rely on human beings for ontology evolution, as very few ontology modellers are both domain and ontology experts. Therefore, automatic ontology evolution is necessary. To address this limitation, they propose a more formal and logic-based method which is based on *belief change* (also known as belief revision) [51; 50; 94]. Belief change deals with the adaptation of a knowledge base (KB) to new information [51]. The authors claim that the focus of belief change is on determining the most rational ways of dealing with changes on knowledge, as well as the development of algorithms that update knowledge base automatically. Viewing an ontology as a special type of KB, they treat ontology evolution as a special case of belief change. Flouris et al. claim that it makes sense to apply the work developed in the belief change literature to the field of ontology evolution. They aim to migrate the belief change theories to the context of ontology evolution. They have sketched an attempt [43; 45] to reformulate the AGM theory[3] [3] in the context of DLs. However, they only studied the feasibility of applying the generalised AGM postulates for belief change to DLs, and the proposed migration of the AGM theory in the ontology evolution context is not complete.

3.2.2 Ontology Versioning

Multi-version reasoning is an approach that tries to cope with possible inconsistencies in changing ontologies by considering both the latest version of an ontology, and all previous versions. It manages the relation between different versions of an ontology, and a notion of compatibility with such versions [65]. Given a number of versions of an ontology, the task of ensuring consistency is reduced to the task of finding the right version of the ontology among all the versions. This

[3]It is a method of belief revision giving minimal properties a revision process should have. The name is from the initials of the authors who established the field – Alchourrón, Gärdenfors and Makinson

task requires the ability to determine the satisfiability of concept descriptions across the different versions of the ontology. (For the details of this approach, we refer the reader to [83].)

3.2.3 Reasoning with Inconsistent Ontologies

Huang et al. [84] claim that it is common to reuse ontologies from other sources. They claim it is impossible to repair those ontologies containing inconsistencies, and the scale of the reused ontologies may be too large to make repairs effective. Therefore, they propose an algorithm reasoning with inconsistency. Their idea is that: given a selection function, which is defined on syntactic relevance, they select some consistent sub-ontology from an inconsistent ontology, then apply standard reasoning on that sub-ontology to obtain meaningful answers.

Example 22 *Given an ontology $\mathcal{O}$ containing the following axioms:*
1. $\mathsf{Bird} \sqsubseteq \mathsf{Animal}$
2. $\mathsf{Bird} \sqsubseteq \mathsf{Fly}$
3. $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$
4. $\mathsf{Penguin} \sqsubseteq \neg\, \mathsf{Fly}$
5. $\mathsf{Tweety{:}Penguin}$
6. $\mathsf{Eagle} \sqsubseteq \mathsf{Bird}$

Tweety *is a* Penguin *which can both fly and not fly. If a query* Tweety:Fly *(can Tweety fly?) is made, the selection function would select those axioms which are directly syntactically relevant to the query (i.e., there is a common concept/role/individual name which appears in both axioms), therefore, a sub-ontology $\mathcal{O}_1$ set by the selection function would contain axioms 2, 4, and 5, as they have common names with* Tweety:Fly. *The inconsistency reasoner will return a negative answer to the query for the sub-ontology, that is, Tweety cannot fly. Formally, $\mathcal{O} \models \neg\, \mathsf{Tweety{:}Fly}$.*

However, the syntactic relevance based selection function does not always provide the intended results, as it always prefers a shorter relevance path. For the mad_cow.owl example[4]. the path 'mad_cow - cow - vegetarian' is shorter than the path 'mad_cow - eat brain - eat body part - sheep are animals - eat animals - NOT vegetarian'. Therefore the syntactic relevance-based selection finds a consistent sub-theory by simply ignoring the fact 'sheep are animals'. As a result, the algorithm returns a true answer for the query 'is mad_cow a vegetarian'. They are still investigating different approaches to selection functions (e.g., semantic-relevance based).

3.2.4 Summary of Approaches to Handling Inconsistency

Overall the above three approaches aim to

1. Ensure the consistency of the ontology when the ontology is changed;

2. Cope with inconsistencies by managing different versions of the ontology;

3. Avoid the inconsistency and apply a non-standard reasoning.

[4]http://www.cs.man.ac.uk/∼horrocks/OWL/Ontologies/mad_cows.owl

In this thesis, we are interested in a fourth alternative: the methods for resolving such inconsistencies. The task of debugging ontologies is getting increasing attention lately, and considerable research effort is being put into the improvement of debugging. In the next section, we focus on describing existing approaches to debugging and repairing ontologies.

3.3 Ontology diagnosis and debugging

The work on diagnosing and debugging inconsistent ontologies is attracting more attention recently. It is generally accepted that ontology users tend to find it difficult to diagnose or debug errors in ontologies. Such problems get worse for those ontologies with numerous unsatisfiable concepts. Many ontology editors (e.g., Protégé [117], SWOOP [92]) and DL reasoners (e.g., FaCT++ [146], Pellet [136]) have been developed to provide support for this task. In this section, we review different techniques for debugging and repairing errors in ontologies. We firstly describe the ontology consistency-checking tools, such as ReTAX+ [99], ReTAX++ [100; 101], Chimaera [109] and ConVisor [16], which focus on detecting errors in taxonomies or ontologies. Then, we discuss three different methodologies for debugging inconsistent ontologies, which are a heuristic approach [154], Maximally Concept-Satisfiable Sub-ontologies (MCSS), and Minimal unsatisfiability-preserving sub-TBoxes (MUPS).

3.3.1 Ontology consistency-checking tools

ReTAX+ [99] is a cooperative taxonomy revision tool. It has a clear set of rules to determine whether a taxonomy is well formed, and an associated set of operators to achieve consistency. As the tool only provides support for building and revising taxonomies, it has been further extended as ReTAX++ [100; 101]. This extended tool provides intuitive graphical interfaces that help knowledge engineers to browse ontologies and resolve the inconsistencies. The user is provided with options to resolve the inconsistencies in the ontology. However, the graph-based approach is not expressive enough for complex applications, because certain complex DL axioms cannot be easily expressed using a graph [40]; for example, one cannot easily express in a graph the axiom:
$\exists R.C \sqcap C \sqsubseteq \forall R.(\exists S.(C \sqcap \neg E)) \sqcup C$.

McGuinness et al. [109] present a tool, Chimaera, which supports ontology merging and diagnosing ontologies. Chimaera includes an analysis capability that allows users to run a diagnostic suite of tests. The output is displayed as an interactive log that allows users to explore the results. Chimaera accepts over 15 designated input format choices (such as Ontolingua [38], Protégé, OKBC-compliant form [27]). Moreover, the tests include incompleteness tests (e.g., missing argument names), syntactic checks (incidence of words (or sub-strings), possible acronym expansion), taxonomic analysis (e.g., redundant super classes, redundant types), and semantic checks (e.g., class definition cycle). Most importantly, these tests are not compatible with RDF, DAML and OWL ontologies (though the authors claimed it would soon be compliant with RDF or OWL); we only focus on logical inconsistency in ontologies in this thesis.

Baclawski et al. [16] aim to formalise the output of consistency-checking tools. They have developed a common ontology in OWL for describing the symptoms of the inconsistencies in OWL documents detected by consistency-checking tools; this ontology is called ‘Symptom Ontology’.

One of the characteristics of this ontology is its hierarchy of symptom classes. Fifteen symptom classes are defined. For example, `DisjointnessFailure` means two disjoint classes have an instance in common. The Symptom Ontology is currently used by the ConsVISor[5]. ConsVISor [15] is a consistency checker for formal ontologies, including both traditional data modelling languages (e.g. UML, KIF) and the more recent ontology languages (e.g. DAML+OIL). ConsVISor makes use of the theorem prover SNARK [140] to detect a logical inconsistency in the axioms for RDF. However, there are some limitations: (1) SNARK may not terminate when an ontology is consistent. (2) ConsVISor uses 'negation as failure' (i.e., closed world assumption) rather than logical negation due to its use of Prolog. That means, ConsVISor implicitly assumes that if two resources have different names and are not explicitly specified to be the same, then they are distinct resources. This assumption to opposite to that in OWL ontology which uses open world assumption. (3) The work focuses on consistency checking of DAML ontologies; it does not repair inconsistencies in ontologies.

3.3.2 A Heuristic Approach

Rector et al. [124] have addressed some common problems of ontology users during modelling OWL ontologies:

- Misunderstanding the difference between equivalence and subsumption relationships
- Misunderstanding the difference global and local restrictions
- Confusing the intersection and union restrictions
- Confusing a negated restriction with a negated relation filler

Based on these common errors, Wang et al. [154] developed a set of *heuristic rules* and incorporated them into a Protégé-OWL[6] plugin. Their program is called OWLDebugger[7]; it can detect commonly occurring error patterns in OWL ontologies. This alleviates the user from troubleshooting the unsatisfiable classes. It helps users track down the reasons for errors in OWL classes. Quasi-natural language explanations for unsatisfiable OWL classes are also generated. In the debugging process, the user is led step by step with explanations of the causes of inconsistency; the conditions that cause the inconsistency are highlighted, for debugging. The authors explain that current DL reasoners modify the originally asserted axioms of the ontology internally in order to optimise the reasoning process, this makes explanation for the unsatisfiability of concepts difficult due to the modified axioms. Their debugger treats the DL reasoner as a 'black box', it is independent of DL reasoners, and so independent of such modifications of axioms. As a heuristic approach, the debugger cannot determine the root cause of unsatisfiability in every case, and is therefore not complete. The authors claim that this limitation has no a serious impact on the usefulness of the debugger. This is because in most cases the mistakes made by ontologists can be described by a small number of error patterns which are adopted in the debugger. As the number of unsatisfiable ontologies available on the Web is quite limited, we cannot confirm the

[5]ConsVISor is available at http://vis.home.mindspring.com

[6]http://protege.stanford.edu/plugins/owl/

[7]http://www.co-ode.org/downloads/owldebugger/

impact of the limitation of their debugger. Currently, the process of resolving inconsistencies is left to the user, and the user has to debug them one by one and run the reasoner frequently to check if consistency has been achieved.

3.3.3 Maximally Concept-satisfiable Sub-ontologies (MCSS)

Baader et al. [11] investigate the problem of finding the maximally satisfiable subsets of ABox assertions; they only consider ABox assertions. Meyer et al. [112] extend this tracing technique to calculate Maximally Concept-Satisfiable Sub-ontologies (MCSSs), in which satisfiable sub-ontologies are obtained by removing just enough axioms to eliminate all errors. A specialised tableau-like algorithm is proposed for finding the maximally concept-satisfiable sub-ontologies of an unfoldable terminology represented in the description logic $\mathcal{ALC}$.

Similarly, Haase and Stojanovic [63] propose an algorithm for finding a consistent sub-ontology by utilising an DL reasoner. The algorithm eliminates a minimal number of axioms that result in a maximal consistent sub-ontology.

The differences of the two approaches are that Meyer et al. modify the tableau algorithm to obtain a set of maximal consistent ontologies by removing problematic axioms, but the algorithm is limited to DL $\mathcal{ALC}$, and optimisation techniques are not considered; Haase and Stojanovic rely on an external optimised DL reasoner to achieve the same result, and the algorithm is independent of the expressivity of DLs. Currently, there is no evidence to show which approach has better performance. In general, both approaches provide the user with a set of maximal consistent subontologies, and leave it up to the user to choose the appropriate sub-ontology, but there is no guideline to select the optimal one. This could be an obvious deficiency if there are numerous such subontologies which all contain large number of axioms. The user might find it difficult to analyse all of the subontologies and choose an optimal one.

3.3.4 Minimal Unsatisfiability Preserving Sub-ontologies (MUPSs)

Schlobach [131] studied ways of explaining incoherencies in the DL $\mathcal{ALC}$. Firstly, the author proposed to pinpoint the so called Minimal Unsatisfiability Preserving Sub-ontologies (MUPSs), which are sets of axioms responsible for an unsatisfiable concept. This is called *axiom pinpointing*. Roughly, a MUPS of a named unsatisfiable concept contains only axioms that are necessary to preserve its unsatisfiability. Removing at least one axiom from the MUPSs of a concept makes it satisfiable. The further detail is given in Section 6.2.1. Secondly, the author applied syntactic generalisation techniques to highlight the exact position of a contradiction within the axioms of the TBox. This is called *concept pinpointing*. Concepts are diagnosed by successive generalisation of axioms until the most general form which is still unsatisfiable is achieved. Schlobach et al. [133] take this approach further and exploit the minimal hitting-set algorithm described by Reiter [126] to calculate *diagnosis sets*, i.e., minimal subsets of an ontology that need to be repaired/removed to make the ontology satisfiable.

Example 23 *Given an ontology $\mathcal{O}$ containing the following axioms:*
1. $A \sqsubseteq C \sqcap D \sqcap E$
2. $C \sqsubseteq \neg D \sqcap F$
3. $G \sqsubseteq C \sqcap \neg F$

Concept A and G are both unsatisfiable, axioms 1 and 2 are pinpointed for the unsatisfiable A (i.e., MUPS(A) = $\{\{1,2\}\}$); axioms 2 and 3 are pinpointed for the unsatisfiable G (i.e., MUPS(G) = $\{\{2,3\}\}$). To resolve the unsatisfiability, at least one axiom from each MUPS of each unsatisfiable concept has to be removed. In this case, we can either remove axiom 2 to resolve two errors, or remove axioms 1 and 3, i.e., the diagnosis set is $\{\{2\}, \{1, 3\}\}$.

The next step is to generalise the pinpointed axioms as much as possible, while still preserving unsatisfiability. For A, the generalised axioms are $A \sqsubseteq C \sqcap D$ and $C \sqsubseteq \neg D$; for G, the generalised axioms are $G \sqsubseteq C \sqcap \neg F$ and $C \sqsubseteq F$.

From the above example, we notice that the concepts in axioms are generalised and only these generalised axioms are shown to the user. However, from the perspective of usability, it is necessary to maintain the correlation between the originally asserted axioms and corresponding generalised axioms. For example, our suggestion is to assign a numerical number to each MUPS, and then tag the concepts which are relevant to unsatisfiability with a superscript, i.e.,
1. $A^1 \sqsubseteq C^1 \sqcap D^1 \sqcap E$
2. $C^{1,2} \sqsubseteq (\neg D)^1 \sqcap F^2$
3. $G^2 \sqsubseteq C^2 \sqcap (\neg F)^2$

We will describe this tagging technique in Chapter 4 where a fine-grained approach is proposed.

Overall, the MCSSs and MUPSs approaches achieve the same result. A MCSS can be obtained by excluding the axioms in any one of its diagnosis sets. Moreover, both approaches are only applicable to unfoldable $\mathcal{ALC}$ terminology, they provide the user with a set of either MCSSs, however, there is no support for selecting the appropriate sub-ontology from the set of MCSSs or removing problematic axioms from the set of MUPSs. To address these limitations, SWOOP [92] is developed to provide more comprehensive support for debugging OWL ontologies. The authors of SWOOP firstly extend the algorithm of MUPSs to support more expressive DL, $\mathcal{SHOIN}(\mathbf{D})$, and provide algorithms of unsatisfiability explanation, and repair unsatisfiability. In the following section, we describe these functionalities in detail.

3.4 SWOOP

SWOOP [92] is a stand-alone ontology editor. It allows users to build and edit ontologies, and to check for errors and inconsistencies (using the reasoner Pellet [136]) in ontologies. Most importantly, explanations are provided to help users understand the cause for (and remove) inconsistencies detected in ontologies. In this section, we review the debugging and repair functionalities provided by SWOOP.

3.4.1 Debugging Unsatisfiable Concepts

Kalyanpur et al. [89] extend the axiom pinpointing technique (i.e., finding MUPS in Section 3.3.4) to the more expressive description logic $\mathcal{SHOIN}$. They utilise a glass-box strategy for finding the first MUPS of an unsatisfiable concept. The description logic tableaux reasoner was modified to keep a trace of the cause for the unsatisfiability of a concept, so that the minimal set of relevant axioms in the ontology that support the concept unsatisfiability was obtained. Their tool, SWOOP, also detects interdependencies between unsatisfiable concepts, in which root and derived unsatisfiable concepts are identified. The user can differentiate the root bugs from others which are caused by the root unsatisfiable concepts, and focus solely on the root concepts. For example, A, an unsatisfiable concept, is the root, and all the subconcepts of A, which are also unsatisfiable, are the derived. If A's unsatisfiability is resolved, then the subconcepts of A become satisfiability. This is a particularly effective approach to fixing a large set of derived unsatisfiable concepts. Then, they use a black-box approach, which is reasoner independent, to derive the remaining MUPSs from the first MUPS. Reiter's Hitting Set Tree (HST) algorithm [126] was adapted to find the remaining MUPSs.

3.4.2 Fine-Grained Justification

Kalyanpur et al. [87] further extended the axiom pinpointing approach to capture *precise justifications*, which determines which parts of the asserted axioms are irrelevant for the unsatisfiability of concepts. Their idea is to rewrite the axioms in ontologies in a normal form and split up conjunctions in the normalised version.

Example 24 *Given an ontology $\mathcal{O}$ containing the following axioms:*
α_1. $A \sqsubseteq \exists R.(C \sqcap D) \sqcap E \sqcap F$
α_2. $C \sqsubseteq \neg D$
α_3. $E \sqsubseteq \neg F$

$A \sqsubseteq \exists R.(C \sqcap D) \sqcap E \sqcap F$ is split into α_{11}: $A \sqsubseteq \exists R.(C \sqcap D)$, α_{12}: $A \sqsubseteq E$ and α_{13}: $A \sqsubseteq F$. α_{11}: $A \sqsubseteq \exists R.(C \sqcap D)$ is further split into α_{111}: $A \sqsubseteq \exists R.K$, α_{112}: $K \sqsubseteq C$, α_{113}: $K \sqsubseteq D$ and α_{114}: $C \sqcap D \sqsubseteq E$. As a result, A has two sets in MUPS: $\{A \sqsubseteq E$, $A \sqsubseteq F$, $E \sqsubseteq \neg F\}$, and $\{A \sqsubseteq \exists R.K$, $K \sqsubseteq C$, $K \sqsubseteq D$, $C \sqsubseteq \neg D\}$. Concept E is removed from the second set, it becomes $\{A \sqsubseteq \exists R.(C \sqcap D), C \sqsubseteq \neg D\}$.

With this technique, SWOOP is able to directly strike out the parts of axioms that do not contribute to the unsatisfiability. For the UI perspective, this would require a correlation between the split axioms and the corresponding asserted axioms. However, the details of the implementation are sketchy in their paper [91].

3.4.3 Ranking Axioms

The above approaches focus on pinpointing the problematic axioms which lead to unsatisfiability of concepts in ontologies, then leave it up to users to resolve the errors. There is no further support for ontology users to fix the detected errors. Kalyanpur et al. [91] go a step further by providing repairing solutions. The authors propose strategies for *ranking* axioms in order of importance, and hence, the highest rank axioms are preserved and the lowest ranked axioms are removed from the ontology. The following strategies are considered to rank ontology axioms:

1. *Arity* of axioms – The arity of an axiom α is the number of unsatisfiable concepts that have at least one MUPS containing α [131]. Some axiom may appear in MUPSs for several different concepts (higher arity value), therefore removing such an axiom could resolve many concepts.

2. *Test cases* – the authors of SWOOP mention that they allow the user to specify test cases describing desired entailments (similar to the idea proposed in [49]). Axioms are then ranked based on the desired entailments they break. This method is most useful for applications with test cases. However, this functionality is not implemented in SWOOP yet.

3. *Provenance information* (about authors, source reliability and timestamp) – The authors of SWOOP mention that the change-log of ontologies can be shared, and that axioms added by a user with a high precedence level will be given high importance. However, the details are sketchy in their paper.

4. Relevance to the ontology in terms of its *usage* – in SWOOP the relevance of an element to the ontology depends on the usage of the element in its ontology. For example, if a concept is heavily instantiated, or if a property is heavily used in the instance data, then changing the axiom definitions of that concept or property is undesirable.

5. *Impact* on an ontology when an axiom is removed or modified – SWOOP ranks axioms based on the number of entailments which would be broken if the axioms were removed. This implies that an axiom which breaks more relationships is more important in the ontology. For now, they consider subsumption/disjointness between atomic concepts and instantiation of atomic concepts as the key entailments to check for when an axiom is removed.

3.4.4 Ontology Repair

SWOOP provides an ontology repair service to generate repair plans to fix errors in an ontology based on the ranking strategies, namely, ‘arity’, ‘impact’ and ‘usage’. Its UI is well designed and implemented. The tool provides a hyperlinked value for each axiom which is calculated based on each strategy. If the user clicks on the hyperlinked value, a window will pop up to explain the obtained value. The total rank for each problematic axiom is the sum of each ranking strategy with a default weight. The weights of these strategies are easily reconfigurable by the user, and so the ranking of axioms is up to the user.

Moreover, the user can preview removal effects before their execution and compare different repair alternatives. When at least one axiom is removed, the fixed and remaining unsatisfiable concepts are listed; the lost and retained entailments are displayed. In some cases, there are `Why?` hyperlinks provided to explain why entailments are lost or retained. This function facilitates the user to debug ontologies in an efficient manner, as the user can quickly preview the impact of all possible removals, and choose the optimal ones. The lost and retained entailments due to removal are often implicit in the ontology, these entailments are not obvious to the user. Thus, it is very useful to inform the user of such implicit information whenever a change is made.

The tool provides suggestions for ontology users rewriting axioms instead of directly removing them from the ontology. A library of commonly occurring error patterns is maintained. If any erroneous axiom has a pattern corresponding to one of the common error patterns, then they suggest the intended axiom to the user as a replacement. The examples of rewriting suggestions include changing intersection to union, changing equivalence to subsumption. Plessers et al. [122] also provide similar suggestions for resolving unsatisfiable concepts. These suggestions could be useful for non-expert users who have no idea how to fix errors. The rewriting suggestions are usually to generalise axioms, this helps reduce the information loss from the ontology, and achieve consistent ontologies.

3.4.5 SWOOP's Scope and Limitations

Overall SWOOP in conjunction with Pellet provides a flexible and interactive framework to debug ontologies. Their usability evaluation also shows that it significantly improves the OWL debugging experience. However, there is still room for improvement.

Firstly, in their ranking approach, the 'arity', 'impact', and 'usage' strategies are currently implemented in the SWOOP editor. As discussed in Chapter 1, it does not necessarily follow that such suggestions give the best results for all ontologies. It is always possible to come up with more heuristics which depend on different factors. To further improve the ranking support, we will analyse the heuristics humans use to solve these kinds of problems in Chapters 5 and 6.

Secondly, it often happens that only parts of axioms cause a concept's unsatisfiability, rather than whole axioms. Even though they provide precise justifications for explaining unsatisfiability, to indicate specific parts of axioms relevant to the unsatisfactorily, they only calculate lost entailments due to removing the whole axioms, but not the problematic *parts* of axioms. Currently, the lost entailments due to removal of axioms are limited to subsumption/disjointness between atomic concepts, and instantiation of atomic concepts.

Lastly, in general, there are numerous possible options to resolve inconsistencies, but usually only very few of these options are intuitively acceptable [105]. In SWOOP, the rewriting axioms suggestions are provided, but these suggestions are limited to a small number of common errors patterns. Moreover, the common error patterns may only apply for those ontologies built by non-expert users, it is insufficient to cover other applications, such as ontology merging/integration. Also, they do not consider the information loss due to their rewriting suggestions. For example, an intersection concept $C \sqcap D$ is changed to be an union $C \sqcup D$, the modified concept is more generic, and hence certain information is lost.

3.5 Heuristic Related Approaches

According to Russell and Norvig [128], a heuristic is a method for problem-solving that ignores whether the solution can be proven to be correct, but which usually produces a good solution. In this section, our aims are to investigate (1) if it is reasonable to utilise humans' heuristics to debug ontologies, and (2) if any work conducts empirical studies to acquire humans' heuristics for debugging ontologies. We review the related approaches, which are divided into three main areas:

1. Think-aloud Protocol Analysis: it is a method used to acquire knowledge from human experts.

2. Empirical Studies in Ontology Engineering: we give an overview of empirical studies reporting actual experiences in developing or deploying ontologies.

3. Heuristics in Ontology Management: we describe a variety of heuristics used in ontology management, which includes ontology mapping, ontology versioning and ontology evolution.

3.5.1 Think-aloud Protocol Analysis

The think-aloud protocol analysis [37] approach aims to elicit the steps a person performs in solving a problem. Using this approach, investigators can identify the information that is concentrated on during problem solving and how that information is used to facilitate problem resolution. From this, inferences can be made about the reasoning processes that were used during the problem-solving task [47]. The purpose of this method is to make explicit what is implicitly present in subjects, who are able to perform a specific task. The think aloud protocol has been applied in psychological and educational research and also for knowledge acquisition in the context of building knowledge-based systems [149], such as AI systems (e.g., [97; 1; 157; 102]), knowledge management tools (e.g., knowledge for software maintenance [4]), etc.

Yu et al. [157] use protocol analysis (i.e., think-aloud method) to obtain both domain knowledge from experts about how to solve problems in a gas turbine and information about how domain experts analyse the archived temporal data. The information helped them to design a prototype knowledge-based system which generates a summary of temporal data in the gas turbine domain.

Laxmisana et al. [102] follow the think-aloud method to analyse the perspectives of personnel involved in decision-making about devices in critical care. The subjects from a hospital were asked to 'think aloud' while evaluating three error scenarios based on real events. The results help them assess the completeness of the problem representations of the subjects, their awareness of critical events, and how these events would collectively contribute to the occurrence of error.

Anquetil et al. [4] followed the think-aloud protocol [103] where the software maintainers were observed performing maintenance; they were asked to explain everything they were doing while maintaining a system, the sessions were taped, the tapes were transcribed. In their study, they identified the kind of knowledge that the software engineers were using at each moment and formalised the knowledge into an ontology. They aimed to build a knowledge management system that could assist in the software maintenance activities (e.g., system investigation, reverse engineering, etc).

The work of Korpi [97] which applied the think-aloud protocol in human concept learning is of particular interest to us. Korpi conducted a psychological study on category induction with the purpose of detecting the strategies and heuristics used by people when coping with puzzling phenomena, that is, phenomena that do not fit within their typical way of understanding. The study used think-aloud protocol elicitation and analysis techniques [37]. The subjects were instructed to report all their thoughts as they solved the categorization task. This procedure allowed subjects to use their natural methods of category induction and provide solutions that were characteristic of their own thinking, without constraints on the types of answers they could give. Korpi detected in her study several strategies that had not been reported in traditional research on concept induction. Korpi's results and investigative approach made an interesting contribution to the study of concept induction.

Alberdi et al. [1; 2] adapted Korpi's empirical approach and tested her model in a real scientific domain with practicing scientists as subjects. They investigated the cognitive processes involved in categorisation tasks performed with conflicting data in the scientific domain of plant taxonomy. Several heuristics which were encountered in their study corroborated Korpi's findings; a novel heuristic was also discovered. The authors claimed that when expert scientists encounter unexpected phenomena in real scientific scenarios, much of their reasoning is focused not only on proposing new experiments but also on generating new hypotheses to explain the data [35]. The authors further reinforced the view that scientists are no longer viewed as possessing special mental abilities; rather, they are viewed as "general" problem solvers who apply their specialized expert knowledge to the solution of scientific problems [39]. Their results suggested that professional scientists can make use of background knowledge to generate new hypotheses when previous explanations prove unfruitful.

In summary, the works of Yu et al., [157] Laxmisana et al. [102] and Anquetil et al. [4] make use of the think aloud protocol to elicit knowledge from experts. The knowledge identified from experts helps them to develop prototype knowledge-based systems. The studies of Kopri [97] and Alberdi [2] have implications both for the study of knowledge-driven categorization, and for the study of scientific reasoning with unexpected observations. We believe that when scientists are faced with surprising observations that contradict their theoretical expectations it is analogous to the situations faced by ontology experts when they encounter ontologies with inconsistencies. Therefore, we are interested in adopting the technique to conduct empirical studies with ontology experts with the purpose of detecting the heuristics used by them when coping with unsatisfiable ontologies.

3.5.2 Empirical Studies: Ontology Engineering

In this subsection we give an overview of the most prominent case studies related to ontologies which have been published in the knowledge or ontology engineering literature. There are number of empirical studies reporting on actual experiences in developing or deploying ontologies. Gómez-Pérez et al. [54] provide more details on this type of empirical study.

Hameed et al. [67] present an approach to acquire knowledge from domain experts and construct ontologies based on multiple experts' knowledge in a uniform way. They have detected a wide array of mismatches among experts' ontologies, extending from simple syntactic discrepancies to a range of rich semantic inconsistencies. They concluded that it is not plausible to evolve an all-encompassing solution. Uschold and Healy [148] conducted an experiment involving reusing and applying an existing ontology in an engineering application. They aimed to identify an ontology reuse situation that is likely to succeed, as well as the kind of technical issues and problems that would need to be faced. Their results reveal important limitations of ontology-driven research: the difficulties of fully automatic translation between any two highly expressive languages, the need for significant effort from the knowledge engineer, and the need for scalable and efficient technologies for knowledge reuse. Russ et al. [127] describe a case study of reusing, merging and translating ontologies. Their results show that translation of ontologies written in different formalisms, and merging several ontologies originally constructed in the same formalism are generally difficult tasks; lots of human intervention is required. They pointed out that an ontological engineering methodology that can provide support and guidance to ontological engineers is necessary.

In summary, the studies usually aim at investigating the methodologies for building ontologies, such as from scratch, or by reusing, integrating, merging. The majority of the experiments have been carried out to validate a particular methodology, or to exemplify the usage of a specific ontology engineering tool. Few studies have applied knowledge acquisition techniques to investigate methodologies for supporting the task of resolving unsatisfiable ontologies (semi-)automatically.

3.5.3 Heuristics in Ontology Management

The task of ontology management even if using conventional editing tools is challenging, time-consuming and error prone. To help ontology modellers to master the complexities of ontology management, several systems and frameworks have been developed to support them to implement their tasks (semi)-automatically. In this section, we review a number of works which use heuristic approaches to alleviate the modeller from labour intensive tasks, which are merging ontologies, ontology versioning and ontology evolution.

3.5.3.1 Merging Ontologies

Ontology merging is a task which compares several (specifications of) ontologies and combines them into a more extensive one [70]. There are several heuristic approaches for supporting the merging of ontologies [81; 143; 118; 116]. Hovy [81] describe several heuristics for identifying corresponding concepts in different ontologies, such as comparing the names and English definition of concepts, matching the closeness of two concepts in hierarchies. Chimaera [110], which is a merging and diagnostic ontology environment, uses a number of heuristics for merging ontologies, such as linguistic heuristics, in which term names, presentation names, term definitions, possible acronyms and suffixes are considered. This is further discussed in Section 3.3.1. The evaluations of the tool suggested that the tool made significant improvements in productivity and quality of ontology development and maintenance. Noy et al. [119] develop an algorithm for ontology merging and alignment – PROMPT, in which [119] linguistic-similarity matches are used as an initial step, and then the structure of the ontology is analysed. Their results showed that PROMPT was very effective in providing suggestions: A human expert followed 90% of

PROMPTs suggestions. PROMPT was also very effective in its coverage: 74% of the knowledge-based operations invoked by the user were suggested initially by PROMPT. Overall, these tools using heuristic-based analysers are proved to facilitate the merging process effectively.

3.5.3.2 Ontology Versioning

Ontology versioning is defined as the ability to handle changes in an ontology by creating and managing different variants of it [96]. As an aid to the task of ontology versioning, certain tools have been developed which automatically identify the differences between ontology versions. PromptDiff [118] is an ontology-versioning system. A heuristic-based algorithm is proposed for comparing ontology versions; it analyses the two versions of ontologies and automatically produces a difference between them (it is called *structural diff*). These heuristics include analysis and comparison of concept names, properties that are attached to concepts, domains and ranges of properties and so on. The authors claim that the heuristic approach finds the changes efficiently and the experimental results show that in practice it also finds the minimal set of changes [116].

3.5.3.3 Ontology Evolution

Ontology evolution is the adaptation of an ontology as changes occur and the consistent propagation of these changes to dependent artefacts [142]. There are several approaches to investigating the problems of changing ontologies. As ontology development is necessarily an iterative and a dynamic process [139], the methods and tools for ontology evolution are required to cope with the changes in a more systematic way. Stojanovic [142] and Haase et al. [64] address the following heuristic approaches in the ontology evolution process:

1. **Structural-driven Change Discovery**
   Structure-driven change discovery is based on the knowledge that ontology engineers use in decision making during ontology evolution. It exploits a set of heuristics to improve an ontology based on the analysis of its structure. Examples of the heuristics include if all subconcepts have the same property, then the property may be moved to the parent concept, or a concept with a single subconcept should be merged with its subconcept, etc. The result of their usability study showed that this approach effectively reduced the user's involvement in an ontology evolution task, because the process is semi-automatic.

2. **Usage-driven Change Discovery**
   Haase et al. [64] exploit heuristics-based techniques to capture and discover the changes in an ontology during ontology evolution. They propose *usage-driven change discovery* which results from the usage patterns created over a period time. Their approach is to capture the users' behavioural patterns, which can in turn be used to reflect changes required for ontology improvement. With this technique, they adapted a collaborative filtering recommender system to assist users in the management and evolution of their personal ontology by providing detailed suggestions of ontology changes. The approach is implemented in the Bibster application [62], which supports the evolution of the users' domain ontology. The evaluation was performed as an experiment with the Bibster community; the result showed the performance improvements over non-personalised recommendations.

3.5.4 Summary of Heuristic Approaches

We have described previous works which use think-aloud protocol to elicit knowledge from humans for problem-solving. Their results shows that humans' heuristics acquired from the studies are useful to build AI tools. The heuristic approaches are also widely used in ontology management as well, for example, humans' heuristics are used in ontology evolution [142] process. A number of empirical studies in ontology engineering are also presented, but no study is about investigating methodologies for supporting the task of resolving unsatisfiable ontologies (semi-)automatically. Therefore, we can conclude two issues: (1) it is reasonable to use humans' heuristics to debug inconsistencies in ontologies; (2) no previous work has conducted empirical studies to acquire humans' heuristics to support users dealing with such inconsistencies.

3.6 Summary and Research Direction

In this chapter, we have reviewed related work which raise our interest in research on debugging ontologies, and gives us a research direction. Before proposing methods for debugging ontologies, we did a thorough review of existing ontology debugging tools, in particular SWOOP. SWOOP provides a number of repair features: identifying (parts of) axioms causing concepts to be unsatisfiable, differentiating root and derived unsatisfiable concepts, ranking problematic axioms and suggesting rewriting axioms. It is undoubted that SWOOP significantly improves the OWL debugging experience. However, we believe that research on debugging ontologies is still in its early stages; the field is a very active, but immature, research area of great importance for the Semantic Web.

Furthermore, we presented a brief literature review on different types of work in an attempt to provide a link between heuristic-based approaches and methods for dealing with inconsistencies in ontologies. We now summarise their work into three points:

1. Think-aloud protocol analysis is useful to acquire humans' heuristics for problem-solving.

2. Most empirical studies on ontological engineering focus on the methodologies of constructing or reusing ontologies, instead of debugging ontologies.

3. Heuristics have been widely used in the task of ontology merging, ontology versioning and ontology evolution.

These works raise an open issue about whether humans' heuristics can be utilised to resolve inconsistencies in ontologies. We, therefore, are interested in acquiring the heuristics humans use to debug ontologies using think-aloud protocol analysis, and then to verify the practical usefulness of these heuristics in an ontology debugging tool.

In this chapter, we have addressed the limitations of existing work on debugging ontologies in order to distinguish between this closely related research and our own research direction. Our research will focus on a heuristic-based approach to ranking problematic axioms and a fine-grained approach to providing the modeller with detailed information about the impact of changes on the ontology.

Chapter 4

Pinpointing Problematic Parts of Axioms

This chapter describes a fine-grained approach to pinpointing problematic parts of axioms which are responsible for an unsatisfiable concept. We extend previous tableau-tracing techniques for debugging by capturing parts of axioms – mainly recording specific concepts that trigger expansion rules as the tableau tree is being built. This technique allows us to rewrite faulty axioms by removing error-causing concepts, instead of removing axioms entirely.

The chapter is organised as follows: Section 4.1 gives the motivation of our tableau-tracing technique. Section 4.2 shows the detail of our proposed approach. Section 4.3 discusses the limitations of the fine-grained approach. Our approach is compared with other related work in Section 4.4. The conclusion of the chapter is given in Section 4.5.

4.1 Motivation

Resolving inconsistencies in ontologies is a challenging task for ontology modellers. Traditional Description Logic (DL) reasoners such as FaCT [75], can check if an ontology is unsatisfiable (i.e., if there are any unsatisfiable concepts in the ontology); however, they do not provide an explanation for the unsatisfiability. Even ontology experts find it difficult to work out why a concept is flagged as unsatisfiable by a reasoner. When faced with several unsatisfiable concepts in a moderately large ontology, the task of debugging can become onerous indeed. Therefore, considerable research effort [131; 112; 87; 91; 122] was put into improvement of debugging.

We have reviewed existing approaches which provide reasons for why concepts in an ontology are unsatisfiable in Chapter 3. Roughly speaking, these approaches mainly focus on how to identify problematic axioms (by providing the minimally unsatisfiable sub-ontologies) [131] or how to directly weaken the target unsatisfiable ontology (by providing the possible maximally satisfiable sub-ontologies) [112]. However practical problems remain: it is not clear which axioms or which parts of axioms should be removed/modified. These approaches have in common that they work at the axiom level and do not identify the specific parts of the axiom responsible for the unsatisfiability.

Example 25 *Let us assume that an ontology $\mathcal{O}$ contains the following axioms:*
α_1*:* $A \doteq C \sqcap \forall R.B \sqcap D$
α_2*:* $C \doteq \exists R.\neg B \sqcap B$
α_3*:* $G \doteq \forall R.(C \sqcap F)$

It can be shown that the concept A is unsatisfiable, by using standard DL TBox reasoning. The most common existing approaches [131; 112] either identify the minimally unsatisfiable sub-ontologies $\mathcal{O}_1^{min} = \{\alpha_1, \alpha_2\}$ or calculate the maximally satisfiable sub-ontologies $\mathcal{O}_1^{max} = \{\alpha_1, \alpha_3\}$, and $\mathcal{O}_2^{max} = \{\alpha_2, \alpha_3\}$. In short, either α_1 or α_2 should be removed from $\mathcal{O}$. However, it is easy to see that we do not need to remove either the whole axiom α_1 or α_2. In order to minimise the loss of information from the ontology, we can simply remove parts of axiom α_1, i.e., (a) $A \sqsubseteq C$, or (b) $A \sqsubseteq \forall R.B$, or part of axiom α_2, i.e., (c) $C \sqsubseteq \exists R.\neg B$, and $\mathcal{O}$ then becomes satisfiable.

In this chapter, we extend Meyer et al.'s tableaux algorithm [112]. Our algorithm not only pinpoints the problematic axioms, but also traces which parts of the axioms are responsible for the unsatisfiability of a target concept A. Currently, only few existing works [131; 87] propose similar fine-grained approaches which identify the faulty parts of axioms. We will compare their work with our approach in Section 4.4, in order to show that our approach is a novel way of achieving the same result as [131; 87].

4.2 A Fine-grained Approach

In this section, we introduce the extended tableau algorithm from Meyer et al.[112] (this kind of tracing technique was first proposed by Baader and Hollunder [11]). Instead of removing whole axioms involved in a concept's unsatisfiability, our algorithm captures the components of axioms responsible for the unsatisfiability.

4.2.1 Extended Tableau Algorithm

As we only consider the TBox $\mathcal{T}$ in an ontology, we use $\mathcal{T}$ to denote an ontology in this chapter. We assume that $\mathcal{T}$ = $\{\alpha_1, \cdots, \alpha_n\}$, with α_i referring to $C_i \doteq D_i$ or $C_i \sqsubseteq D_i$ for $i = 1, \ldots, n$. We divide $\mathcal{T}$ into an unfoldable part $\mathcal{T}_u$ and a general part $\mathcal{T}_g$, such that $\mathcal{T}_g$ = $\mathcal{T} \setminus \mathcal{T}_u$.

Tableaux algorithms test the satisfiability of a concept D by constructing a representation of a model for it – an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ in which $D^{\mathcal{I}} \neq \emptyset$. A model of D is usually represented by a tree **T**. Each node x in the tree is labelled with a set $\mathcal{L}(x)$ of concept or role elements. The concept elements are of the form $(a : C, I, a' : C')$, where C and C' are concepts, a and a' are individual names, and I is an index-set. This means that the individual a belongs to concept C due to an application of an expansion rule on $a' : C'$. The set of axioms, which $a : C$ comes from, is recorded in the index-set I. This is done by adding i to I, which is a set of integers in the range 1, $\ldots, n$. In an element of the form $(a : C, I, a' : C')$ we frequently refer to C as "the concept", and a as "the individual"(i.e., we are referring to the first concept assertion). When a concept element $(a : C, I, a' : C')$ is in the label of a node x, it represents an interpretation $\mathcal{I}$ that satisfies C, i.e., the individual corresponding to a is in the interpretation of C. Formally, if

$(a : C, -, -) \in \mathcal{L}(x)$, then $a \in C^{\mathcal{I}}$, where "$-$" stands for any value, that is, it is a place holder.

Role elements are of the form $(R(a, b), I, a : \exists R.D)$, where R is a binary relationship between individuals a and b; I is the index-set; the third parameter is to record the existence of $R(a, b)$ due to an application of an expansion rule on $a : \exists R.D$. Formally, if $(R(a, b), -, -) \in \mathcal{L}(x)$, then $\langle a, b\rangle \in R^{\mathcal{I}}$.

4.2.2 Applications of Expansion Rules

To determine the satisfiability of a concept D in $\mathcal{T}$, a tableau algorithm initialises a tree **T** to contain a single node x, called the root node, with $\mathcal{L}(x) = \{(a : D, \emptyset, nil)\}$. The tree is then expanded by repeatedly applying a set of expansion rules which either extend node labels or add new leaf nodes. Our extended set of expansion rules for the $\mathcal{ALC}$ Description Logic is shown in Table 4.1, where A_i is a concept name, C, C_1, C_2, C_i, D_i are concept descriptions, R is a role name, a and b are individual names, $RHS(\alpha_i)$ is the concept at the right-hand side of α_i, and the signature $Sig(\alpha_i)$ of an axiom α_i is the set of concept and role names occurring in α_i.

$U^{+}_{\doteq}$-rule	if $A_i \doteq C_i \in \mathcal{T}_u$, $(a : A_i, I, -) \in \mathcal{L}(x)$ and $(a : C_i, I \cup \{i\}, a : A_i) \notin \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_i, I \cup \{i\}, a : A_i)\}$
$U^{-}_{\doteq}$-rule	if $A_i \doteq C_i \in \mathcal{T}_u$, $(a : \neg A_i, I, -) \in \mathcal{L}(x)$ and $(a : \neg C_i, I \cup \{i\}, a : \neg A_i) \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \neg C_i, I \cup \{i\}, a : \neg A_i)\}$
$U_{\sqsubseteq}$-rule	if $A_i \sqsubseteq C_i \in \mathcal{T}_u$, $(a : A_i, I, -) \in \mathcal{L}(x)$ and $(a : C_i, I \cup \{i\}, a : A_i) \notin \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_i, I \cup \{i\}, a : A_i)\}$
$\sqcap$-rule	if $(a : C_1 \sqcap C_2, I, -) \in \mathcal{L}(x)$, and $\{(a : C_1, I, a : C_1 \sqcap C_2), (a : C_2, I, a : C_1 \sqcap C_2)\} \nsubseteq \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_1, I, a : C_1 \sqcap C_2), (a : C_2, I, a : C_1 \sqcap C_2)\}$
$\sqcup$-rule	if $(a : C_1 \sqcup C_2, I, -) \in \mathcal{L}(x)$, and $\{(a : C_1, I, a : C_1 \sqcup C_2), (a : C_2, I, a : C_1 \sqcup C_2)\} \cap \mathcal{L}(x) = \emptyset$, then create two $\sqcup$-successor y, z of x with: $\mathcal{L}(y) := \mathcal{L}(x) \cup \{(a : C_1, I, a : C_1 \sqcup C_2)\}$ $\mathcal{L}(z) := \mathcal{L}(x) \cup \{(a : C_2, I, a : C_1 \sqcup C_2)\}$
$\exists$-rule	if $(a : \exists R.C, I, -) \in \mathcal{L}(x)$, a is not blocked (see Section 4.2.4), and $\{(R(a, b), I, a : \exists R.C), (b : C, I, a : \exists R.C)\} \nsubseteq \mathcal{L}(x)$, where b is an individual name not occurring in $\mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(R(a, b), I, a : \exists R.C), (b : C, I, a : \exists R.C)\}$
$\forall$-rule	if $(a : \forall R.C, I, -) \in \mathcal{L}(a)$, and $(R(a, b), J, a : \exists R.D_i) \in \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(b : C, I \cup J, a : \forall R.C)\}$
$\sqsubseteq$-rule	if $(C_i \sqsubseteq D_i) \in \mathcal{T}_g$, and there exists $(- : E, -, -) \in \mathcal{L}(x)$, $Sig(E) \cup Sig(C_i) \neq \emptyset$, a is not blocked, and $(a : \neg C_i \sqcup D_i, I \cup \{i\}, -) \notin \mathcal{L}(x)$, for every individual a in the node then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \neg C_i \sqcup D_i, I \cup \{i\}, a : C)\}$

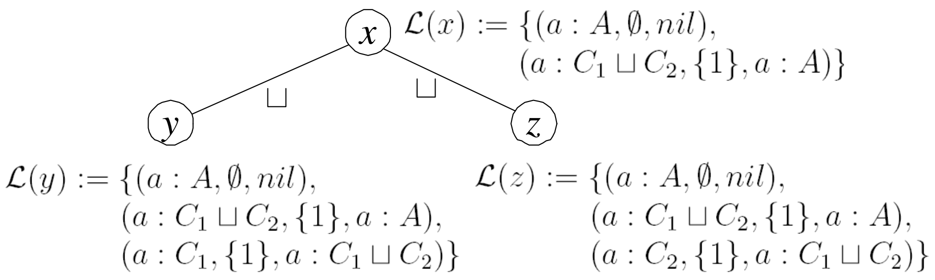
Table 4.1: Our extended tableaux expansion rules for $\mathcal{ALC}$

Same as the original tableau algorithm (as described in Section 2.3.1), during the expansion, concept descriptions are assumed to be converted to negation normal form. We now explain the extended expansion rules. The three rules ($U^{+}_{\doteq}$-rule, $U^{-}_{\doteq}$-rule and $U_{\sqsubseteq}$-rule) describe the *unfolding* procedure (as described in Section 2.3.1).

Disjunctive concept elements $(a : C_1 \sqcup C_2, -, -) \in \mathcal{L}(x)$ result in *non-deterministic* expansion. We deal with this non-determinism by creating two $\sqcup$-successors y, z of x with:

$\mathcal{L}(y) := \mathcal{L}(x) \cup \{(a : C_1, \cdots)\}$, and $\mathcal{L}(z) := \mathcal{L}(x) \cup \{(a : C_2, \cdots)\}$.

Example 26 *Given a single axiom: $A \sqsubseteq C_1 \sqcup C_2$, to test the satisfiability of A, the algorithm creates a node x whose label is initialised as: $\mathcal{L}(x) := \{(a : A, \emptyset, nil)\}$, where a is an individual name, it is an instance of A. The $U_{\sqsubseteq}$-rule is then applied to the axiom, it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_1 \sqcup C_2, \{1\}, a : A)\}$, that means, a is also an instance of $C_1 \sqcup C_2$. Next, the $\sqcup$-rule is applied to $(a : C_1 \sqcup C_2, \{1\}, a : A)$, two $\sqcup$-successors, y, z, are created. The label of nodes y and z is initialised to be that of node x, i.e., $\mathcal{L}(y) := \mathcal{L}(x)$ and $\mathcal{L}(z) := \mathcal{L}(x)$. To represent that a is either an instance of C_1 or C_2, a new element $(a : C_1, \{1\}, a : C_1 \sqcup C_2)$ is added to the label of $\mathcal{L}(y)$; $(a : C_2, \{1\}, a : C_1 \sqcup C_2)$ is added to the label of $\mathcal{L}(z)$. (see Figure 4.1)*



x
$\mathcal{L}(x) := \{(a : A, \emptyset, nil), (a : C_1 \sqcup C_2, \{1\}, a : A)\}$
$\sqcup$
$\sqcup$
y
z
$\mathcal{L}(y) := \{(a : A, \emptyset, nil), (a : C_1 \sqcup C_2, \{1\}, a : A), (a : C_1, \{1\}, a : C_1 \sqcup C_2)\}$
$\mathcal{L}(z) := \{(a : A, \emptyset, nil), (a : C_1 \sqcup C_2, \{1\}, a : A), (a : C_2, \{1\}, a : C_1 \sqcup C_2)\}$

Figure 4.1: The fully expanded tree with two leaf nodes y, z

For any existential role restriction concept $(a : \exists R.C, I, -) \in \mathcal{L}(x)$, the algorithm introduces a new individual b as the role filler, and this individual must satisfy the constraints expressed by the restriction. Thus, b is an individual of C, and hence $(b : C, I, a : \exists R.C)$ and $(R(a, b), I, a : \exists R.C)$ are added to the label of the node. A universal role restriction concept $(a : \forall R.D, J, -) \in \mathcal{L}(x)$ interacts with already defined role relationships to impose new constraints on individuals. That is, if $(R(a, b), I, a : \exists R.C)$ exists in $\mathcal{L}(x)$, then b is also an individual of D; a new concept element $(b : D, I \cup J, a : \forall R.D)$ is added to the label.

If there exists a concept C in the signature of the left-hand side of a GCI axiom ($\alpha_i \in \mathcal{T}_g$, α_i is $C_i \sqsubseteq D_i$), and there is an element $(a : C, I, -) \in \mathcal{L}(x)$, and the signature of C has common elements with $Sig(C_i)$ then we apply the $\sqsubseteq$-rule to α_i. The newly added element will be $(a : \neg C_i \sqcup D_i, I \cup \{i\}, a : C)$. With this technique we are able to trace which element in the tree invokes the application of expansion rules on GCI axioms, therefore we can trace how the GCI axioms cause the concept's unsatisfiability.

The algorithm repeatedly expands the tree by applying the rules in Table 4.1 as many times as possible until either any one of the fully expanded leaf nodes has no clash or none of the rules is applicable to any node of the tree. A node is fully expanded when none of the rules can be applied to it. **T** is fully expanded when all of its leaf nodes are fully expanded. A node x contains an obvious *clash* when, for some individual b and some concept C, $\{(b : C, -, -), (b : \neg C, -, -)\} \subseteq \mathcal{L}(x)$.

When a clash is found in a node, the classical tableau algorithm [11] either backtracks and selects a different leaf node, or reports the clash and terminates, if no node remains to be expanded. The main difference is that our algorithm terminates when either (1) any one of the fully expanded leaf nodes is without a clash or (2) none of the rules is applicable to any node of the tree. Since the rules are still applicable to a node even when a clash is found, there may be more than one clash in the node, and furthermore this clash may also occur in other nodes (repeated nodes). As a result, we can obtain all the clashes in the tree and eliminate the repeated clashes. If the input of the tableaux algorithm is a concept C and a terminology $\mathcal{T}$, we have the following property: C is *unsatisfiable* iff each path from the root to the leaf node in the tree contains at least one clash. This implies that an unsatisfiable concept becomes satisfiable if all the clashes in any one of the paths of the tree are resolved (i.e., a complete path from root to leaf). This is because whenever the non-deterministic $\sqcup$-rule is applied, two new $\sqcup$-successor nodes are created; this is the only way to create the leaf nodes. It is sufficient to resolve all clashes in either of the two branches created.

4.2.3 Sequences of a Clash

Definition 1 (Sequences of a Clash) *Given a clash in a tree, the sequences of a clash, Seq^+ and Seq^-, contain elements involved in the clash. The sequences are of the form $\langle (a_0 : C_0, I_0, a_1 : C_1), (a_1 : C_1, I_1, a_2 : C_2), \cdots, (a_{n-1} : C_{n-1}, I_n, a_n : C_n), (a_n : C_n, \emptyset, nil)\rangle$, where $I_{i-1} \subseteq I_i$ for each $i = 1, \cdots, n$. The first elements of Seq^+ and Seq^- are of the form $(a : C, I', a' : C')$ and $(a : \neg C, I'', a'' : C'')$ respectively. The last element of both sequences is the same.*

(1) Initialise the root node x with $\mathcal{L}(x) := \{(a : A, \emptyset, nil)\}$,

(2) Apply the $U^+_{\sqsubseteq}$-rule to $(a : A, \emptyset, nil)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A)\}$,

(3) Apply the $\sqcap$-rule twice to $(a : C \sqcap \forall R.B \sqcap D, \{1\}, \cdots)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D)$,
$(a : D, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : C, \{1\}, a : C \sqcap \forall R.B)$,
$(a : \forall R.B, \{1\}, a : C \sqcap \forall R.B)\}$

(4) Apply the $U^+_{\sqsubseteq}$-rule to $(a : C, \{1\}, \cdots)$, followed by applying the $\sqcap$-rule,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C)$,
$(a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B), (a : B, \{1, 2\}, a : \exists R.\neg B \sqcap B)\}$,

(5) Apply the $\exists$-rule to $(a : \exists R.\neg B, \{1, 2\}, \cdots)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(b : \neg B, \{1, 2\}, a : \exists R.\neg B), (R(a, b), \{1, 2\}, a : \exists R.\neg B)\}$

(6) Apply the $\forall$-rule to $(a : \forall R.B, \{1\}, \cdots)$,
it gives $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(b : B, \{1, 2\}, a : \forall R.B)\}$

Figure 4.2: The application of expansion rules on A in Example 25

Figure 4.2 shows how the tableau algorithm is applied to Example 25 (shown in Section 4.1) to check for the satisfiability of A. The tree **T** contains a node x which has a clash because $\{(b : B, \{1, 2\}, a : \forall R.B), (b : \neg B, \{1, 2\}, a : \exists R.\neg B)\} \subseteq \mathcal{L}(x)$. According to Definition 1, we can obtain two sequences, Seq^+ and Seq^- (see Figure 4.3). Note that the union of the index sets of the first elements in the sequences of the clashes in the tree gives the set of axioms which cause A to be unsatisfiable. The above two sequences show that axiom α_1 and α_2 cause the unsatisfiability of A.

$$
\begin{aligned}
\enclose{circle}{x}\ \mathcal{L}(x) := \{&(a : A, \emptyset, nil), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),\\
&(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : D, \{1\}, a : C \sqcap \forall R.B \sqcap D),\\
&(a : C, \{1\}, a : C \sqcap \forall R.B), (a : \forall R.B, \{1\}, a : C \sqcap \forall R.B),\\
&(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C), (a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B),\\
&(a : B, \{1, 2\}, a : \exists R.\neg B \sqcap B), (b : \neg B, \{1, 2\}, a : \exists R.\neg B),\\
&(R(a, b), \{1, 2\}, a : \exists R.\neg B), (b : B, \{1, 2\}, a : \forall R.B)\}
\end{aligned}
$$

$$
\begin{aligned}
Seq^{+} := \langle&(b : B, \{1, 2\}, a : \forall R.B), (a : \forall R.B, \{1\}, a : C \sqcap \forall R.B),\\
&(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),\\
&(a : A, \emptyset, nil)\rangle,
\end{aligned}
$$

$$
\begin{aligned}
Seq^{-} := \langle&(b : \neg B, \{1, 2\}, a : \exists R.\neg B), (a : \exists R.\neg B, \{1, 2\}, a : \exists R.\neg B \sqcap B),\\
&(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C), (a : C, \{1\}, a : C \sqcap \forall R.B),\\
&(a : C \sqcap \forall R.B, \{1\}, a : a : C \sqcap \forall R.B \sqcap D), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),\\
&(a : A, \emptyset, nil)\rangle
\end{aligned}
$$

Figure 4.3: The fully expanded tree for A in Example 25

Now that clashes in the tree have been found, we want to identify the axioms which could be removed to resolve the unsatisfiability. We can identify these by looking at the nodes in the tree which contain clashes. The example in Figure 4.4 (on the left-hand side) shows a tree with six clashes (nodes with clashes are shaded). We can see various ways to resolve the unsatisfiability; firstly the clashes in node a definitely must be resolved, as all the other nodes in the tree descend from it; secondly the clashes in one of the nodes b or c must be resolved, as all the other nodes in the tree descend from one of these; thirdly the clashes in one of the nodes b or e or f must be resolved; finally the clashes in one of the nodes b or e or d must be resolved. Therefore we have four sets: $\{a\}, \{b, c\}, \{b, e, f\}, \{b, e, d\}$; we must pick one node from each set and resolve its clashes. For example, resolving the clashes in a and b would be sufficient to remove the unsatisfiability.

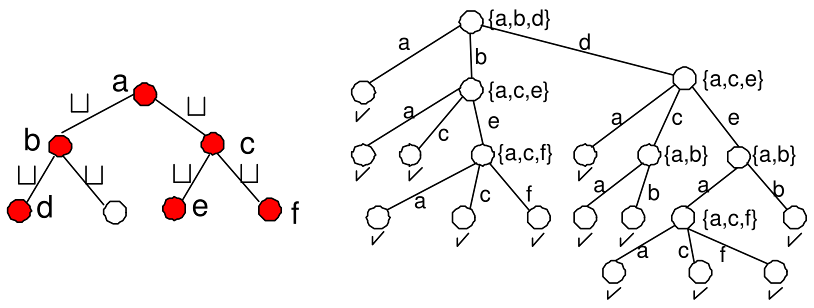


Figure 4.4: Left-hand side: a fully expanded tree with six clashes; right-hand side: hitting set tree

We now describe how Reiter's Hitting Set algorithm [126] can be adapted to make a general procedure for identifying these sets. Firstly, for each path from the root to a leaf of the tree, we gather the set of each of the nodes on that path which has a clash. In our example the following sets are found: $\{a, b, d\}, \{a, b\}, \{a, c, e\}, \{a, c, f\}$. Now, using these sets we apply the Hitting Set algorithm; the Hitting set Tree is shown in Figure 4.4 on the right-hand side. Now for each leaf node n we gather the set E_n of all edge labels on the path from the root to that node. The sets thus obtained from each leaf node are gathered into one large set S. This gives a set with 13 elements; some of these $E_n \in S$ are subsets of each other; we pick out the *minimal* sets; i.e., the sets $E_i \in S$ for which there is no $E_j \in S$ such that $E_j \subset E_i$. In our example this gives $S = \{\{a\}, \{b, c\}, \{b, e, f\}, \{b, e, d\}\}$, as desired. The axioms involved in each clash from the above nodes are actually the same as the notion of minimal unsatisfiability preserving sub-TBoxes (MUPS) in [131], that is there are four sets of MUPS in the unsatisfiable concept above.

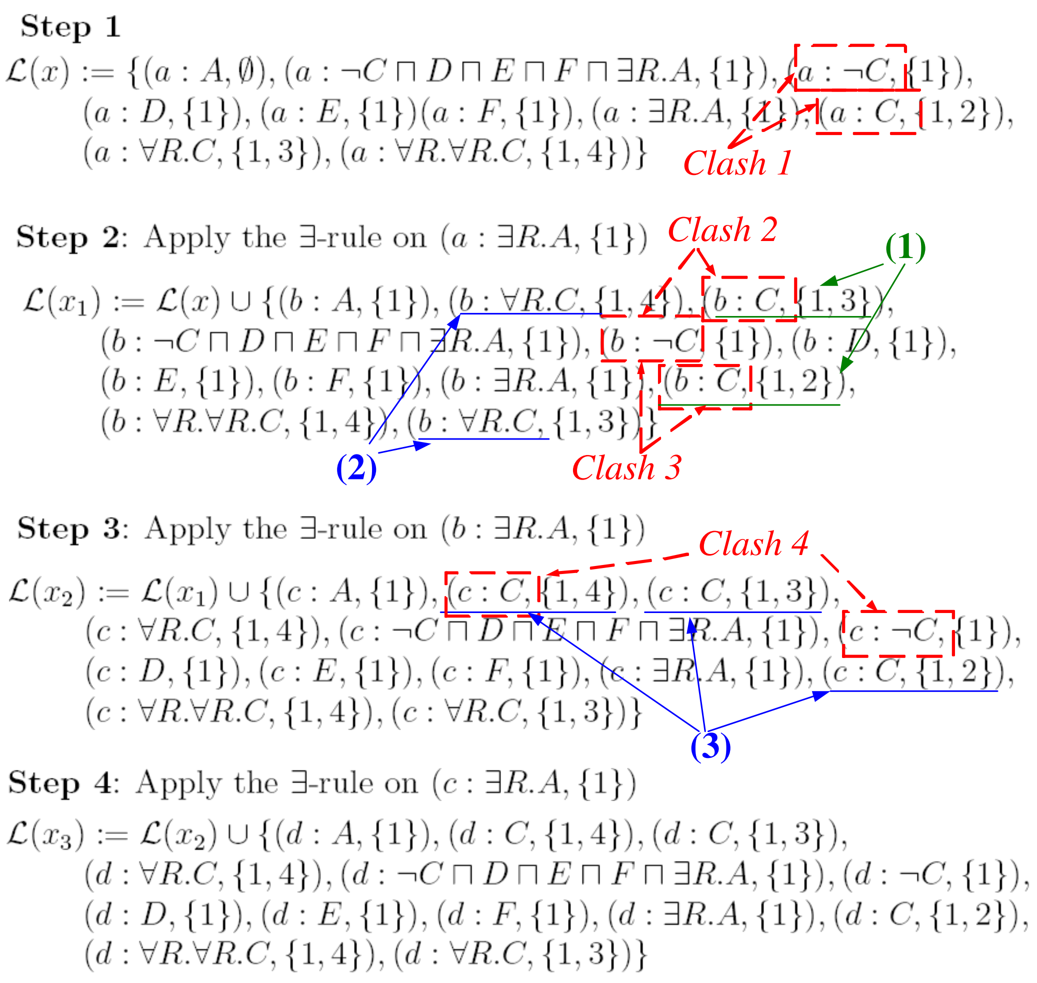
From each MUPS, we know which axioms cause the unsatisfiability. Furthermore, from the sequences of the clashes, we know which concepts within these axioms cause the unsatisfiability. We can assign a specific number to each MUPS, and annotate the problematic concepts in these axioms with a specific superscript number corresponding to the MUPS which it occurs in. Note that a concept component may be involved in more than one MUPS, therefore it may be annotated with more than one number. We introduce the notion of arity of a concept C in an axiom α, denoted by $arity(\alpha, C)$, to count the number of times it appears in the clashes. This idea is similar to the core of MUPS in [131]. This means that removing a concept component with arity n can resolve n clashes. In order to illustrate the benefit of our fine-grained approach, we add the following axioms to Example 25:
α_4: $K \doteq C \sqcap \forall R.(P \sqcap F)$
α_5: $P \doteq \forall R.F \sqcap B$

In this case, concept K is also unsatisfiable due to the existence of a clash in a node of the tree for K. For simplicity, we do not show the sequences in the clash. We now annotate the concepts in the axioms which are involved in the two unsatisfiable concepts with superscript numbers as follows:
α_1: $A^1 \doteq C^1 \sqcap \forall R^1.B^1 \sqcap D$,
α_2: $C^{1,2} \doteq \exists R^{1,2}.(\neg B)^{1,2} \sqcap B$,
α_3: $G \doteq \forall R.(C \sqcap F)$
α_4: $K^2 \doteq C^2 \sqcap \forall R^2.(P^2 \sqcap F)$
α_5: $P^2 \doteq \forall R.F \sqcap B^2$

From above, we can easily see which concepts in the axioms cause which concepts to be unsatisfiable. It is obvious that removing concept $\exists R.\neg B$ in axiom α_2 can resolve two unsatisfiable concepts.

4.2.4 Refined Blocking

To deal with cyclic axioms, it is necessary to add cycle detection (often called *blocking*) to the preconditions of some of the expansion rules in order to guarantee termination [8; 24]. Our blocking approach is slightly different from the classical one (described in Section 2.3.2). We define the refined blocking condition as follows: the application of the $\exists$-rule and the $\sqsubseteq$-rule to an individual a is blocked by b iff $\{(D, I)|(a : D, I, -) \in \mathcal{L}(x)\} = \{(D', I')|(b : D', I', -) \in \mathcal{L}(x)\}$. Informally, the justification for this refinement is the following. If I is not equal to I', then we treat $(a : C, I, -)$ and $(a : C, I', -)$ as different elements; this is because the concept C in the two elements has been introduced from different axioms. Therefore, we still apply the rules to both $(a : C, I, -)$ and $(a : C, I', -)$ to expand the tree. As a result, in our approach, an individual a is blocked by b iff each of the elements in $\mathcal{L}(x)$ with individual a is exactly matched with one of



Step 1

$\mathcal{L}(x) := \{(a : A, \emptyset), (a : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (a : \neg C, \{1\}),$
$(a : D, \{1\}), (a : E, \{1\})(a : F, \{1\}), (a : \exists R.A, \{1\}), (a : C, \{1, 2\}),$
$(a : \forall R.C, \{1, 3\}), (a : \forall R.\forall R.C, \{1, 4\})\}$

Step 2: Apply the $\exists$-rule on $(a : \exists R.A, \{1\})$

$\mathcal{L}(x_1) := \mathcal{L}(x) \cup \{(b : A, \{1\}), (b : \forall R.C, \{1, 4\}), (b : C, \{1, 3\}),$
$(b : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (b : \neg C, \{1\}), (b : D, \{1\}),$
$(b : E, \{1\}), (b : F, \{1\}), (b : \exists R.A, \{1\}), (b : C, \{1, 2\}),$
$(b : \forall R.\forall R.C, \{1, 4\}), (b : \forall R.C, \{1, 3\})\}$

Step 3: Apply the $\exists$-rule on $(b : \exists R.A, \{1\})$

$\mathcal{L}(x_2) := \mathcal{L}(x_1) \cup \{(c : A, \{1\}), (c : C, \{1, 4\}), (c : C, \{1, 3\}),$
$(c : \forall R.C, \{1, 4\}), (c : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (c : \neg C, \{1\}),$
$(c : D, \{1\}), (c : E, \{1\}), (c : F, \{1\}), (c : \exists R.A, \{1\}), (c : C, \{1, 2\}),$
$(c : \forall R.\forall R.C, \{1, 4\}), (c : \forall R.C, \{1, 3\})\}$

Step 4: Apply the $\exists$-rule on $(c : \exists R.A, \{1\})$

$\mathcal{L}(x_3) := \mathcal{L}(x_2) \cup \{(d : A, \{1\}), (d : C, \{1, 4\}), (d : C, \{1, 3\}),$
$(d : \forall R.C, \{1, 4\}), (d : \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A, \{1\}), (d : \neg C, \{1\}),$
$(d : D, \{1\}), (d : E, \{1\}), (d : F, \{1\}), (d : \exists R.A, \{1\}), (d : C, \{1, 2\}),$
$(d : \forall R.\forall R.C, \{1, 4\}), (d : \forall R.C, \{1, 3\})\}$

Figure 4.5: The fully expanded tree for A in Example 27

the elements in $\mathcal{L}(x)$ with individual b, and vice versa; i.e., these matched elements have the same concept and index-set. In contrast, the classical tableau algorithm does not take the index-set of axioms into account when blocking is performed; the elements in the labels of nodes only have one parameter. The elements $(a : C, I, -)$ and $(a : C, I', -)$ will be presented as $(a : C)$ in the classical one, and therefore only one rule is applied to $(a : C)$ once.

Example 27 *This example describes how the refined blocking works:*
α_1: $A \sqsubseteq \neg C \sqcap D \sqcap E \sqcap F \sqcap \exists R.A$
α_2: $D \sqsubseteq C$
α_3: $E \sqsubseteq \forall R.C$
α_4: $F \sqsubseteq \forall R.\forall R.C$

We use Example 27 to illustrate why our refined blocking is necessary.[1] For simplicity, we do not show the third parameter of the elements in the node label. Figure 4.5 shows the fully expanded tree. In step 2, after applying the $\exists$-rule on $(a : \exists R.A, \{1\})$, we can see that, in $\mathcal{L}(x_1)$ in the classical algorithm, the set of concept elements with b is a subset of the set of concept elements with a, therefore, individual b is blocked by a. The $\exists$-rule cannot be applied on $(b : \exists R.A, \{1\})$,

[1]Note that despite the apparent complexity, this example is simplest possible in order to illustrate the need for our refined blocking.

so the algorithm would terminate. Three clashes are found in $\mathcal{L}(x_1)$; axioms α_1, α_2 and α_3 are all involved in the clashes (cf. *Clash 1, 2, 3* in Figure 4.5). However α_4 would be missed out by the classical algorithm, although it also triggers a clash in our tree. The reason is that the set of concept elements with a is the same as the set of concept elements with b in the classical algorithm (i.e., $\{D|(a : D, -) \in \mathcal{L}(x_1)\}$ = $\{D|(b : D, -) \in \mathcal{L}(x_1)\}$); $(b : C, \{1, 3\})$ is the same as $(b : C, \{1, 2\})$ (cf. **1** in Figure 4.5), and $(b : \forall R.C, \{1, 4\})$ is the same as $(b : \forall R.C, \{1, 3\})$ (cf. **2**). In our approach, these elements are different. Therefore we still apply the $\forall$-rule to $(b : \forall R.C, \{1, 4\})$ and add a new element $(c : C, \{1, 4\})$ into the node label, which is different from the existing elements $(c : C, \{1, 2\})$ and $(c : C, \{1, 3\})$ (cf. **3**). The newly added element from α_4 triggers another clash (cf. *Clash 4*). Next, we keep applying the $\exists$-rule to $(c : \exists R.A, \{1\})$ and create a new individual d. Finally, each of the elements in $\mathcal{L}(x_3)$ with individual d is exactly matched with one of the elements in $\mathcal{L}(x_3)$ with individual c, and vice versa; i.e., these matched elements have the same concept and index-set. The individual c is blocked by d, and then the application of the rules is terminated.

4.2.5 Complexity, Soundness and Completeness

The differences between our algorithm and the classical one are that (1) when a clash is detected, the classical algorithm either backtracks and selects a different node, or reports the clash and terminates if no more nodes remain to be expanded, whereas our algorithm will not do so; it only terminates when the tree is fully expanded or until blocking occurs, in order to find all possible clashes. Therefore, the complexity of our algorithm is the same as the classical one in the worst case [134], as both need to fully expand all nodes; (2) we add two extra parameters in each of the elements of a node label. The expansion rules do not depend on these two parameters, and hence they add only a constant amount to each expansion and do not affect the complexity and correctness of the original algorithm [11]; (3) our refined blocking condition is: the application of the $\exists$-rule and the $\sqsubseteq$-rule to an individual a is blocked by b iff $\{(D, I)|(a : D, I, -) \in \mathcal{L}(x)\} = \{(D', I')|(b : D', I', -) \in \mathcal{L}(x)\}$. The number of elements with different concept descriptions that can be introduced in each fully expanded leaf node is finite. Also, for each of concept description C, there can be only a finite number of elements $(a : C, I_1, -)$, $(a : C, I_2, -)$,$\cdots$, $(a : C, I_n, -)$ with n bounded by the number of axioms in the ontology. The algorithm is therefore guaranteed to terminate.

4.2.6 Removing clashes

Given an unsatisfiable concept A in $\mathcal{T}$, we can obtain a fully expanded tree containing a node with at least one clash. For each clash, the sequences of the clash, Seq^+ and Seq^-, are obtained as in Definition 1. We can derive the following lemma:

Lemma 2 *Let the first elements of the sequences be $(a : C, I', -)$ and $(a : \neg C, I'', -)$, and let the last element of the sequences be $(b : A, \emptyset, nil)$. We know that the set of axioms $I := I' \cup I''$ causes A to be unsatisfiable. Let $\mathcal{D}$ be the set of all concepts appearing in the elements of the sequences, removing one of the concepts in $\mathcal{D}$ from one of the axioms in I is sufficient to resolve the clash.*

Proof For any concept picked from $\mathcal{D}$, it must occur in the sequences and have an adjacent element which is before or after. For any two adjacent elements in a sequence, e_1 and e_2, there are only two possibilities:

- e_1 and e_2 are of the form $(a : E_1, -, a : E_2)$ and $(a : E_2, -, -)$ containing the same individual, this means the concept E_1 is a superconcept of E_2. If E_1 (or E_2) is removed, the subsumption relationship between E_2 and E_1 is removed. Therefore, the individual a no longer belongs to E_1 (or E_2), nor does it belong to any of the concepts in the elements preceding the occurrence of e_1 in the sequences. That means the concept of the first element in the sequence is not subsumed by the removed concept either, hence the clash is resolved.

- e_1 and e_2 are of the form $(a : E_1, -, b : E_2)$ and $(b : E_2, -, -)$ containing different individuals, this means the concept E_1 participates in a role relationship with E_2. If E_1 or E_2 is removed, then the role relationship will be removed, therefore there will be no such individual a participating in the role, and all the concepts in the elements preceding the occurrence of e_1 will not be related to a, and hence the clash will be resolved.

∎

4.3 Discussion

In this section, we discuss the limitations of our fine-grained approach, which currently supports $\mathcal{ALC}$; a few optimisation techniques are applicable to the approach.

4.3.1 More Expressive DLs

Currently, the fine-grained approach only supports terminologies in $\mathcal{ALC}$ with GCIs. In order to increase the applicability of the approach, it is necessary to extend the algorithm to support the Description Logic $\mathcal{SHOIN}(\mathbf{D})$ [79] as an OWL-DL ontology corresponds to a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base. For example, in DL $\mathcal{ALCN}$ [71], in order to detect inconsistencies due to conflicting numbers restrictions, we need to add a new type of clash: for some node x, $\{(a :\leq nR, -, -)\} \cup \{(R(a, y_i), -, -) | 1 \leq i \leq n+1\} \cup \{y_i \neq y_j | 1 \leq i < j \leq n+1\} \subseteq \mathcal{L}(x)$ for individual names a, y_1, …, y_{n+1}, a non-negative integer n, and a role name R.

For this type of clash, our sequences of a clash are not longer applicable. We need to extend our technique to support more expressive DLs in future work.

4.3.2 Optimisations

The non-determinism in the expansion rule (i.e., $\sqcup$-rule) results in poor performance of the tableau algorithm. Existing DL reasoners, Fact++ [146], Racer [60] and Pellet [136], employ optimisation techniques. They have demonstrated that even with expressive DLs, highly optimised implementations can provide acceptable performance in realistic DL applications. For example, dependency directed backtracking is used to prune the search tree. However, in our fine-grained approach, these techniques are no longer applicable, except for absorption and caching, because we aim to detect all possible clashes by fully expanding the tree. This will adversely affect the performance of the algorithm especially if there is extensive non-determinism in the ontology.

4.3.2.1 Absorption

Most importantly, the absorption technique [74] is still applicable in our revised algorithm, because it is algorithm independent. Absorption is used to preprocess the ontology before the tableau algorithm is applied. For example, given two axioms (1) $\mathsf{CN} \sqsubseteq \forall R.\neg C \sqcap \neg D$, (2) $\forall R.\neg C \sqsubseteq \neg$ CN $\sqcup D$. Axiom (2) will be absorbed as $\mathsf{CN} \sqsubseteq D \sqcup \exists R.C$, and can then be merged with axiom (1), the resulting axiom is $\mathsf{CN} \sqsubseteq (\forall R.\neg C \sqcap \neg D) \sqcap (D \sqcup \neg \exists R.C)$. We then apply the fine-grained algorithm to trace which parts of the axiom cause the unsatisfiability. In this example, with absorption, two axioms are modified into one axiom, we can only tag the parts of the resulting axioms (modified by the absorption) relevant to the unsatisfiability, instead of the original asserted axioms. To improve usability, further work may be necessary to explain the correlation between the originally asserted axioms and the resulting axioms (modified by the absorption) in an understandable way to the user.

4.3.2.2 Caching

The caching technique [74] is to use cached results from previous tableaux tests to demonstrate the satisfiability of a concept. This technique can save significant computation time, and it is applicable in our approach.

Example 28 *Given an ontologies consisting of the following axioms:*
1. $C \sqsubseteq F \sqcap E$
2. $F \sqsubseteq \neg E$
3. $A \sqsubseteq C \sqcap D$

To test the satisfiability of C*, we apply the expansion rules to axioms 1 and 2, and obtain the following fully expanded tree with a root node* x*:*
$\mathcal{L}(x) := \{(a : C, \{\}, nil), (a : F \sqcap E, \{1\}, a : C), (a : F, \{1\}, a : F \sqcap E),$
$(a : E, \{1\}, a : F \sqcap E), (a : \neg E, \{1, 2\}, a : F)\}$

Therefore, C *is unsatisfiable; the above result is cached. We now test the satisfiability of* A*, the unfolding and the* $\sqcap$*-rules are applied to axiom 3, the following result is obtained:*
$\mathcal{L}(y) := \{(b : A, \{\}, nil), (b : C \sqcap D, \{3\}, b : A), (b : C, \{3\}, b : C \sqcap D), (b : D, \{3\}, b : C \sqcap D)\}$

The next step is to apply the unfolding rule to $(b : C, \{3\}, b : C \sqcap D)$*. We can skip the application of this rule by making use of the cached result from* C*, and merge the elements in* $\mathcal{L}(x)$ *with that in* $\mathcal{L}(y)$*. The individuals and index-sets of elements from* $\mathcal{L}(x)$ *have to be updated accordingly. The following merged result is obtained:*
$\mathcal{L}(y) := \{(b : A, \{\}, nil), (b : C \sqcap D, \{3\}, b : A), (b : C, \{3\}, b : C \sqcap D),$
$(b : D, \{3\}, b : C \sqcap D), (b : F \sqcap E, \{1, 3\}, b : C), (b : F, \{1, 3\}, b : F \sqcap E),$
$(b : E, \{1, 3\}, b : F \sqcap E), (b : \neg E, \{1, 2, 3\}, b : F)\}$

We will further describe this caching technique in Section 7.4.

4.4 Related Work

In this section we compare our approach with two related fine-grained approaches used in debugging ontologies.

Schlobach et al. [131] apply syntactic generalisation techniques to highlight the exact position of a contradiction within the axioms of the TBox. This is called *concept pinpointing*. Concepts are diagnosed by successive generalisation of axioms until the most general form which is still unsatisfiable is achieved. The main difference with our work is that in [131] the concepts in axioms are generalised and only these generalised axioms are shown to the user. For example, α_1: $A \sqsubseteq C \sqcap D \sqcap E$, α_2: $C \sqsubseteq \neg D \sqcap F$, then the generalised axioms $A \sqsubseteq C \sqcap D$, and $C \sqsubseteq \neg D$ are shown. It can be an additional burden on the user to correlate between the generalised axioms and the originally asserted axioms. In the case of very complicated axioms, the user might find it difficult to know which generalised axioms correspond to which of the original asserted axioms. Compared to our approach, we use a tracing technique to pinpoint problematic parts of axioms. The technique is able to indicate that a concept component is responsible for more than one concept's unsatisfiability. Continue with the example, we add a new axiom α_3: $G \sqsubseteq C \sqcap \neg F$, with our technique, the problematic parts are tagged as follows:
α_1: $A^1 \sqsubseteq C^1 \sqcap D^1 \sqcap E$,
α_2: $C^{1,2} \sqsubseteq (\neg D)^1 \sqcap F^2$,
α_3: $G^2 \sqsubseteq C^2 \sqcap (\neg F)^2$

Kalyanpur et al. [87] also propose a fine-grained approach, which determines which parts of the asserted axioms are responsible for the unsatisfiability of concepts. Their idea is to rewrite the axioms in an ontology in a normal form and split up conjunctions in the normalised version, e.g., $A \sqsubseteq \exists R.(C \sqcap D)$ is rewritten as $A \sqsubseteq \exists R.E$, $E \sqsubseteq C$, $E \sqsubseteq D$ and $C \sqcap D \sqsubseteq E$. In comparison, we achieve the same results as their approach, but we identify irrelevant parts of axioms by making use of the tableau algorithm, instead of splitting the axioms.

Furthermore, Kalyanpur et al. [87] use a black-box approach, which is reasoner independent, to derive the remaining MUPSs from the first MUPS, as an alternative to fully expand the tableau tree. Reiter's Hitting Set Tree (HST) algorithm [126] was adapted to find the remaining MUPSs [87]. They form an HST where nodes correspond to a single MUPS, edges correspond to the removal of axioms from the ontology, and in building the tree to compute all minimal hitting sets, they in turn obtain all the MUPSs as well. The advantage of this approach is that it makes use of the optimisation techniques embedded in the reasoner. The disadvantage is that the complexity of generating the HST is exponential with the number of MUPSs. They argued that their results showed the algorithm performed well in practice, because most of the satisfiability tests exhibited at most three or four MUPSs, with five to ten axioms each [86]. In comparison, we detect all MUPSs of an unsatisfiable concepts by fully expanding the tree. The disadvantage of our approach is that most optimisations are disabled; however, its complexity is independent of the number of MUPSs. The practical use and efficiency of our algorithm is demonstrated in Section 8.3.

4.5 Conclusion

We presented a fine-grained approach to identify faulty parts of axioms, instead of whole axioms. Our algorithm is able to find all possible clashes in a tableau tree by fully expanding the tree. We have also discussed the limitations of our technique and compared with related work, and found that we achieve the similar result with the related work by using a tracing technique. However, we have to emphasize that our technique is not just limited to find faulty parts of axioms, but also provides the modeller with detailed information about the impact of changes on the ontology. Specifically, this technique provides a foundation for identifying changes which are helpful in that they restore lost entailments (due to axiom removal), and those that are harmful in that they cause additional unsatisfiability. The details will be covered in Chapter 7. The practical feasibility of the algorithm is evaluated in Chapter 8.

After identifying and understanding the reasons for the concepts' unsatisfiability, the next step is to act, that means to select an appropriate axiom for removal/modification to resolve the unsatisfiability. In the next two chapters, we will describe the empirical studies in which humans' heuristics were acquired, then we provide a heuristic-based service to recommend to the user which axiom(s) should be selected for removal/modification.

Chapter 5

Humans' Heuristics to Resolve Inconsistencies in Ontologies

This chapter introduces our empirical studies to investigate the heuristics used by ontology engineers to resolve unsatisfiable ontologies. In the studies, we have acquired a variety of heuristics based on such things as the impact of removal on the ontologies, knowledge of ontology structures, common OWL modelling errors, etc. We compare these with the heuristics incorporated in existing ontology debugging tools and find that while some are common, we have also discovered new useful heuristics. Interestingly, our studies also reveal some contradictory heuristics. For example, some ontology engineers believe that longer axioms could be clumsy and complex, therefore, they are likely to be erroneous, and should be removed; however, some claim that longer axioms carry more information, and hence should be preserved to minimise the information lost. We have gained insights into ways in which apparently contradictory heuristics could be made useful if provenance information or information about the process by which the ontology is constructed can also be made available. The practical usefulness of these heuristics in an ontology debugging tool will be presented in Chapter 8.

The chapter is organised as follows: Section 5.1 presents the motivation of our empirical studies; Section 5.2 describes the details and results of the empirical studies; Section 5.3 presents the related work; Section 5.4 gives the conclusion of the chapter.

5.1 Motivation

Earlier work has produced systems such as SWOOP [92] and OWLDebugger [154], which are able to identify the problematic axioms for unsatisfiable concepts in ontologies. Give an unsatisfiable concept, the systems can pinpoint the minimal sets of axioms causing the concept to be unsatisfiable (also known as MUPSs[1]). Removing or modifying at least one of axioms from each set can resolve the unsatisfiability. However, it will be a challenging task for non-expert ontology users to select appropriate axioms for removal or modification. This is because, nowadays, ontologies available on the Semantic Web are getting more complicated (i.e., richly axiomatised, not just simple concept hierarchies), and contain domain specific terminologies. Non-expert users

[1]MUPSs stands for minimal unsatisfiability-preserving sub-TBoxes, which is originally proposed by [131]. The detail is given in Chapter 6.

may find it difficult to understand the logical formalisms or domain specific terminologies in ontologies, e.g., the Gene Ontology [137], GRAIL [123] and TAMBIS [17] ontologies. On the other hand, there is no obvious way for a tool to select correct or appropriate axioms for modification, because it requires an understanding of the meaning of the concepts in the ontologies. A common approach is to leave it up to the user to determine the optimal way to resolve such errors.

It is uncontroversial that the task of ontology management without tool support is difficult, labour intensive and error prone. Many systems and frameworks for supporting the knowledge engineer in ontology management tasks (such as versioning, merging, alignment) have been developed. As a result of reviewing the literature, we find that the heuristic approaches, which are derived from the behaviour of ontology engineers when confronted with the task of managing ontologies (i.e., human behaviour is simulated), are useful to help ontology modellers perform their tasks (semi)-automatically. For example, merging two existing ontologies manually is challenging and time-consuming even when using conventional ontology editing tools. To address this problem, Stumme et al. [143], Noy et al. [118] and Hovy [81] propose syntactic and semantic matching heuristics to offer extensive merging support. Similarly, ontology evolution is not a trivial process, due to the variety of sources and consequences of changes. Therefore, Stojanovic [142] exploits a set of heuristics to improve an ontology based on the analysis of its structure in the ontology evolution process. This has been discussed in Section 3.5.3.

The task of debugging ontologies is very similar to that of ontology management. We believe that it will be useful to acquire the expertise and experience of ontology engineers in resolving inconsistencies in ontologies, and then encode this knowledge as *heuristics* into an ontology debugging tool, which can provides guidance on how to select appropriate axioms for removal/modification. There are many possible definitions of heuristics; we adopt the following, modified slightly, from Russell and Norvig [128]:

> . . . a technique designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem.

For example, ontology experts may believe that sibling classes are usually disjoint with each other. We then suggest the non-expert modeller not to remove disjoint axioms which exists between sibling classes when they are involved in inconsistencies.

5.2 Empirical Studies

In this section, we present the empirical studies approach – *think-aloud protocol*. We firstly describe the experimental design, and then give the details of the studies.

5.2.1 Think-aloud Protocol

Recall that Section 3.5.1 suggests that the think-aloud protocol is useful to acquire knowledge from experts. In particular, the work of Kopri [97] and Alberdi et al. [1; 2] raised our interest in applying similar techniques to debugging ontologies. We believe that when scientists are faced with surprising observations that contradict their theoretical expectations it is analogous to situations faced by ontology experts when they encounter ontologies with inconsistencies. In this chapter, we describe the process of empirical studies with ontology experts; we aim to acquire the heuristics used by them when faced with unsatisfiable ontologies.

A series of empirical studies were conducted to identify the heuristics used by ontology engineers when facing unsatisfiable ontologies. We focused on investigating the situations where ontology engineers encounter ontologies with modelling errors, and note how they interpreted the errors and resolved them. Given the complex and knowledge-intensive nature of the task, and the relatively little research conducted previously on similar issues, an *exploratory* investigative approach, similar to the one used by Korpi and Alberdi et al., has been used. That is, we have followed a methodology close to protocol analysis [37] where we have asked subjects to *think aloud* as they worked through particular tasks; additionally, the investigator also asked the subjects additional questions if she thought this would clarify how a particular subject was addressing a task. The essential issue about exploratory empirical studies is that when one starts such a study one does not have a clear hypothesis in mind, but from the way the subjects address the tasks and the subsequent questioning of the subject, one then formulates hypotheses about how each subject addresses the task. Such hypotheses, once formulated, can then be tested with a larger number of subjects conventionally, i.e., where the results would be statistically evaluated.

5.2.2 Experimental Design

As presented in Section 3.1, in the literature, three notions of ontology consistency are distinguished, namely structural, logical and user-defined consistency [63]. In the studies, we only focus on the logical consistency of OWL ontologies; that means, the ontology is logically inconsistent if it contains contradictory information. We categorise inconsistencies into three types [90]:

- Complement – An individual belongs to a class and its complement. (Task 1 in Appendix A.1 is an example)
- Cardinality – An individual has a maximal cardinality restriction but is related to more distinct individuals. (Task 3 in Appendix A.1 is an example)
- Datatype – A literal value violates the (global or local) range restrictions on a datatype property. (Task 5 in Appendix A.1 is an example)

There are a number of other factors which we believe would greatly affect the knowledge engineer's ability to resolve inconsistencies in ontologies, these include:

- Whether the examples incorporate concrete (English words) or abstract (anonymous) terms in ontologies
- The way in which the examples (often referred to in experimental psychology as the stimuli) are presented. In this domain possible presentation formats are: natural language statements, description logic axioms, and graphical representation

5.2.3 Overview of the Studies

Three studies with subjects with different background and different types of stimuli were conducted. At the beginning of each session with each subject, a warm-up task was conducted to ensure that the instructions were clear. The details of the studies are given in Appendices A, B, and C.

In the first phase, two subjects without extensive knowledge of DLs volunteered for the study. The concept and role names in the ontologies were displayed as concrete (English words) and abstract (anonymous) terms respectively; the stimuli contained inconsistencies due to complement, cardinality, and datatype. From the results of the first phase, we found that it was hard to obtain consistent outcomes as the subjects tended to remove DL axioms randomly and without any specific reasons.

In the second phase, subjects with extensive knowledge of DLs and OWL ontologies were chosen. We noticed that we could not draw any general conclusion from the subjects doing concrete examples as they drew on their personal knowledge of the domain; on the other hand, with abstract stimuli, the subjects considered the impact of changes on ontologies and the structure of the ontologies.

Finally, in the third phase, we focused on abstract stimuli. We further proposed five factors, which depended on the size of axioms, the structure of ontologies and impact of removal on ontologies and usage of classes. Ten subjects who are experienced in DLs and modelling OWL ontologies volunteered to participate in our study. The results show that some heuristics used by the subjects are very similar, while some could be somewhat contradictory.

We now describe the results, general comments and conclusions for each phase of the study in the following sections.

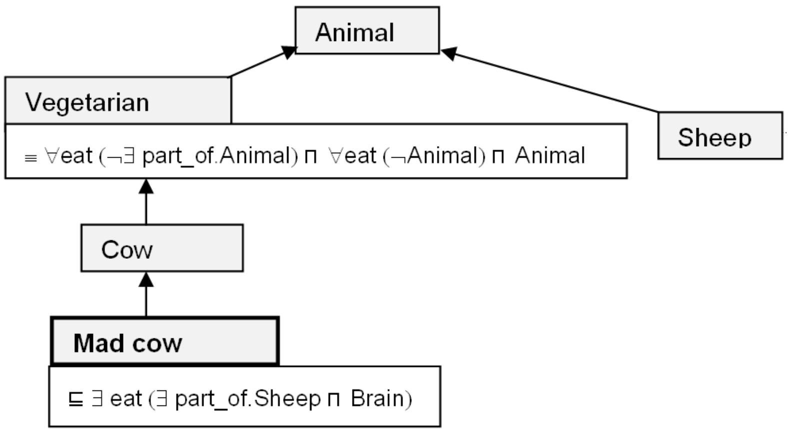
5.2.4 Phase One: Pilot Study with Non-Experts in DLs

In the first study, two subjects from our research group took part in the study; both subjects had no detailed knowledge of DLs but had some experiences with ontologies. The worksheets prepared for the two subjects both contained the identical eight stimuli, where four incorporated only concrete concepts and four involved only abstract concept names. The first worksheet presented the concrete and abstract stimuli alternately; the second worksheet presented all the concrete tasks followed by all the abstract tasks. Both worksheets contained warm-up tasks as a way of explaining to the subjects, the tasks they were being asked to undertake. Figure 5.1 shows a concrete example. The actual worksheets used in this study are in Appendix A.

5.2.4.1 Results

When faced with abstract stimuli, the subjects were not sure what changes should be made; they failed to give justification why certain change was made. In order to finish the task, they could only make a random choice, such as removing one of the axioms randomly. In contrast, with concrete stimuli, we can summarise the strategies used by the subjects as follows:

1. Removing axioms – The most common strategy was to remove the disjointness axioms or subclass-of links. The reason is that all the classes were linked by subclass-of or disjointness


Animal
Vegetarian
≡ ∀eat (¬∃ part_of.Animal) ⊓ ∀eat (¬Animal) ⊓ Animal
Sheep
Cow
Mad cow
⊑ ∃ eat (∃ part_of.Sheep ⊓ Brain)

Mad_cow is unsatisfiable, because it is a vegetarian which does not eat any part of animals, but mad cow itself eats sheep's brain.

Problematic Axioms:
1. Mad_cow $\sqsubseteq$ $\exists$ eat.(($\exists$ part_of.Sheep) $\sqcap$ Brain) $\sqcap$ Cow
[Mad cows are cows that eat the brains of sheep]
2. Cow $\sqsubseteq$ Vegetarian
[Cows are vegetarians]
3. Vegetarian $\doteq$ ($\forall$ eat.($\neg\exists$ part_of.Animal))$\sqcap$($\forall$ eat.($\neg$ Animal)) $\sqcap$ Animal
[Vegetarians do not eat anything that is a part of an animal]
4. Sheep $\sqsubseteq$ Animal
[Sheeps are animals]

Figure 5.1: One concrete stimulus from the first phase of the empirical study with three types of presentational format.

links in the graphs; the subjects focused on these links to solve problems, rather than looking at the definitions of classes. They explained that removing links is easier and quicker than changing the definitions of classes.

2. Splitting a class into two classes – If the ontology example contained an exception of a classification, the subjects would split a class into two in order to accommodate an exceptional subclass. For the typical Mad_Cow example, mad_cow is an exception of cow which is not vegetarian; one of the subjects split the class vegetarian into vegetarian and non_vegetarian, therefore, cow belonged to vegetarian and mad_cow belonged to non_vegetarian.

3. Generalisation

   (a) Changing the parent of a class to a higher level of the hierarchy – If the definition of a class had a conflict with its parent, the subjects would change its parent to a higher level of the hierarchy. For the Mad_Cow example, one of the subjects changed the parent the Mad_Cow to be Animal instead of Cow.

   (b) Changing an intersection to union – If a conflict resulted from an intersection of two or more classes, the subjects would often change the intersection to be union, but there was no specific reason given for why a particular change was made rather than another.

5.2.4.2 General Comments

Resolving inconsistencies was considered to be very difficult by these subjects. The individual sessions took between 45 and 50 minutes. When facing inconsistent 'abstract' ontologies, the subjects could not apply their commonsense knowledge to choose the best way to resolve the inconsistency. Therefore, the subjects were not sure of the correctness of their proposed changes, and they tended to remove DL axioms randomly and without any specific reasons. They appreciated that, often, there was no single correct answer, and often they quickly selected an action which would remove the particular inconsistency. It appears to the experimenter that they did not consider the impact of the changes on the quality of the resulting ontologies.

5.2.4.3 Conclusions

A number of strategies, such as splitting classes, when subjects were presented with the concrete stimuli, were noted. However, with abstract stimuli, the subjects changed ontologies to remove inconsistencies without any apparent reasons (e.g., they removed axioms randomly), and even when asked why other options were not considered, they claimed any changes, which could resolve the problem, were the same. Therefore we decided that the next study should be conducted with subjects who have a more extensive knowledge of DLs. We also planned to be more persistent in our questioning about the reasons why certain changes were made by the subjects; the subjects would be encouraged to give details and reasons for their decisions.

5.2.5 Phase Two: Experts in DLs

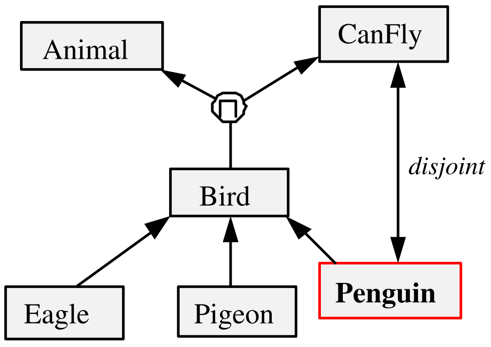
As often happens with experimental studies this first pilot study gave us indeterminate results. We concluded that we needed to run a further study with subjects who are more knowledgeable about DLs. In this phase two subjects from our department were chosen to do the study. We used the same worksheet with both subjects which comprised of four 'concrete' tasks and one 'abstract' task. Figure 5.2 shows two sample stimuli in the task. The actual worksheet used is available in Appendix B.

5.2.5.1 Results

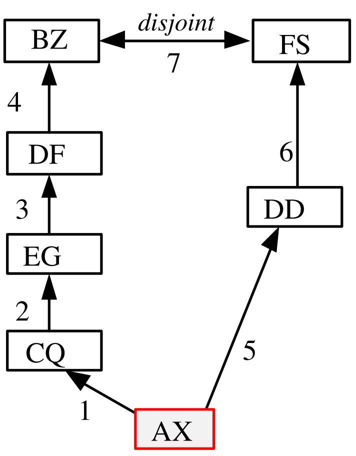
We report two issues from this study:

1. The subjects used their background knowledge to resolve inconsistencies in concrete examples. They changed ontologies by deleting definitions, moving definitions to other classes, splitting a class etc. For the example shown in Figure 5.2 at the left-hand side, Bird can have two subclasses, CanFly and CannotFly, then Eagle and Pigeon are the subclasses of CanFly; Penguin is the subclass of CannotFly. These strategies are similar to the results in phase one.

2. The subjects considered the impact of changes on the ontology in the abstract examples. As they could not apply any domain knowledge, they assumed every axiom in the ontology was modelled correctly. Therefore, they tried to remove a particular axiom to minimise the



Problematic Axioms:
1. Bird ⊑Animal ⊓ CanFly
[Birds are animals which can fly]
2. Penguin ⊑ Bird
[Penguins are birds]
3. Penguin ⊑ ¬ CanFly
[Penguins cannot fly]



Problematic Axioms:
1. AX ⊑CQ
2. CQ ⊑ EG
3. EG ⊑ DF
4. DF ⊑ BZ
5. AX ⊑ DD
6. DD ⊑ FS
7. FS ⊑ ¬ BZ

Figure 5.2: One concrete stimulus (left-hand side) and one abstract stimulus (right-hand side) from the second phase of the empirical study

information loss from the ontology. Both subjects believed that the most specialised axioms gave more information; therefore they would keep more specific axioms. For the example shown in Figure 5.2 (at the right-hand side), the subjects would either remove axiom 1 or 5, because the class AX has four ancestors along axiom 1; it has two ancestors along axiom 5, therefore, axiom 1 carries more information, and hence axiom 5 should be removed. One subject further explained that another option would be to remove the disjoint axiom, so that the class AX could be the subclasses of six ancestors and retain all the properties and information from its ancestors.

5.2.5.2 General Comments

In this study, it seems the abstract example is more useful for acquiring subjects' heuristics. They considered different aspects, such as information loss, impact of changes on the ontology, specialisation etc. When considering the concrete tasks, the subjects applied their commonsense knowledge to resolve the problems; they did not consider the impact of changes and information loss on the resulting ontologies. Effectively, they modelled the domain themselves and made a comparison between their own model and the stimuli presented. They then rewrote the axioms to match their own model. Both subjects achieved similar results and appeared to use similar strategies:

1. In the concrete examples: Both believed that they did not have any special heuristics for resolving the examples; they simply applied their commonsense knowledge to judge inconsistencies and possible changes.

2. In the abstract example: The first subject considered how to minimise the change on the ontology. He analysed how much information would be lost if a certain axiom were removed. Drawing on his knowledge of default logics, he believed more specialised axioms are more accurate, and should override the more general ones. The second subject tried to change the ontology as little as possible. He tried to find axioms which, if removed, would involve least information loss.

Both the two subjects in the second study were knowledgeable about logics and OWL ontologies, they were able to give the experimenter reasons why they made certain changes, and not others.

5.2.5.3 Conclusions

We draw two conclusions from this experiment. Firstly, in the concrete examples the subjects made extensive use of their commonsense knowledge and their domain knowledge to resolve inconsistencies; they changed ontologies by deleting definitions, moving definitions to other classes, splitting a class etc. in order to match their own idea of what needed to be modelled. We have so far not been able to extract generic heuristics from these responses. Secondly, in the abstract example the subjects considered the impact of changes on the ontology. As they could not apply their commonsense knowledge, they tried to remove certain axioms to minimise the information loss from the ontology. They evaluated the lost information by the number of subsumption relations lost due to axiom removal.

5.2.6 Phase Three: Experts in DLs with abstract stimuli only

In the previous study, the worksheet contained both concrete and abstract examples; however, only the problematic axioms in the ontologies were shown to the subjects. In this study, for the reasons given in the conclusions of the last study, we presented only abstract examples. Moreover, whole ontologies were shown to the subjects in this study as it was felt it would give subjects more background, but at the same time the problematic axioms were highlighted for each task. Additionally following the feedback from subjects in the last study and as a result of reviewing the literature, we planned to investigate the following factors:

1. Size of axioms[2] – Do subjects prefer changing/deleting smaller or larger sized axioms?

2. Disjointness between sibling classes – Do subjects believe sibling classes should usually be disjoint with each other? How do disjoint axioms affect the changes subjects make to the ontology?

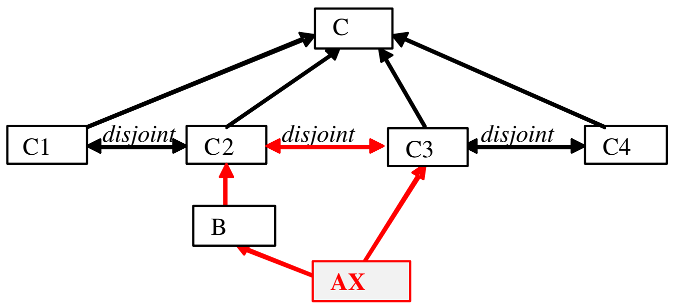
3. Patterns of sibling classes – Do subjects believe the siblings of classes should usually participate in similar relationships/patterns?

4. Hierarchy level of classes – How do subjects judge the speciality and generality of classes in a hierarchy?

[2]The size of an axiom is defined as the total number of class names and role names in the axiom.

5. Usage of classes - Is an axiom more important if the usage of the axiom is higher (i.e., the axiom is highly referenced by other axioms) in the ontology? This criterion is borrowed from SWOOP which uses a 'usage' factor to measure the importance of axioms in an ontology [91]. (The usage of a class relates to its connectivity.)

Ten subjects, familiar with DLs and OWL ontologies, agreed to do this experiment. The subjects were given the same set of stimuli which consisted of five abstract stimuli. The first stimulus was presented only as a set of DL axioms so that we could investigate if the subjects would concentrate on the issue of axiom size. The remaining stimuli were presented in both DLs and graphical format to help the subjects understand the ontologies easily. Figure 5.3 shows a sample stimulus. The actual worksheet used is in Appendix C.



AX is unsatisfiable because the instances of AX belong to class C2 and C3 simultaneously, but C2 and C3 are disjoint with each other.
Problematic Axioms:
1. AX $\sqsubseteq$ B
2. AX $\sqsubseteq$ C3
3. B $\sqsubseteq$ C2
4. C3 $\sqsubseteq \neg$ C2

Figure 5.3: One abstract stimulus from the third phase of the empirical study

5.2.6.1 Results

Tables 5.1 and 5.2 show the results obtained from the eight subjects for each factor. The results of the two subjects are not shown in the tables, because they did not consider the five factors in the study. We summarised the subjects' strategies for resolving inconsistencies in terms of the five factors identified as follows:

1. Size of Axioms – Some subjects believed longer axioms are more clumsy and likely to be erroneous, and so they should be removed, however, some claimed that longer axioms carry more information, and hence are more important and should be retained to minimise the loss of information.

2. Disjointness between sibling classes – Four subjects thought that sibling classes should be disjoint with each other. Two subjects believed that disjointness is unlikely to be asserted by mistake, as ontology modellers usually introduce them purposefully; however, one subject said that non-expert ontology modellers usually do not understand disjointness. For the sample stimulus shown in Figure 5.3, most subjects would keep the disjoint axiom, because

it was important or it exists between two siblings. One subject guessed the ontology modeller tried to make a complete partition of the sibling classes, that means all sibling classes are disjoint with each other, but that the modeller actually forgot to add disjoint axioms between C1 and C3, C1 and C4, C2 and C4; further, the subject suggested that the modeller might mistakenly think that disjointness was transitive. Some subjects further explained that most disjoint classes were close by (‘close by’ means the classes have short distance to their common ancestors), such as sibling classes, the distance between disjoint classes was usually very small (as their direct common parents are the same). Also, it is seldom that disjoint classes are at different levels of the hierarchy.

3. Pattern of sibling classes – The stimulus is shown in Figure 5.4. Five subjects agreed that sibling classes should have similar patterns/properties. One subject claimed that ontology modellers usually misunderstood universal restrictions. For an example from [153], Human $\doteq$ $\forall$ hasParent.Human; it is obvious for the user that everything whose parents are humans must be a human as well, but by using the universal restriction and the equivalent axiom it is implicit that any instance in the ontology which has no parents will also be classified as a human. Another three subjects focused on the correctness of the universal restriction of a class, instead of the sibling pattern. These three subjects regarded universal restrictions as too ‘strong’, because either none or all of the instances must fulfil the relation fillers. One of them explained such strong modelling, which is too restrictive, is less likely to be true in the real world; however, two of them believed that ontology modellers introduce such ‘strong’ restrictions only after careful consideration and when they are confident.

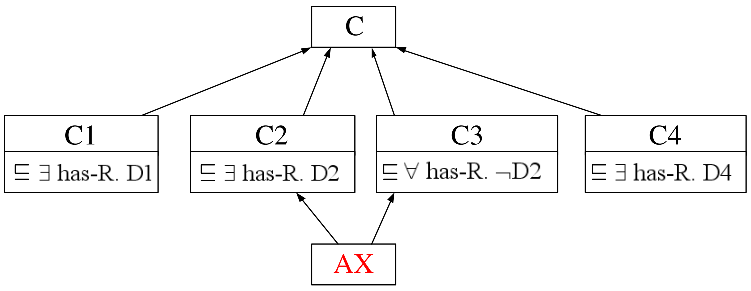


Figure 5.4: One stimulus illustrates the pattern of sibling classes

4. Hierarchy level of classes – There are four cases to be considered here:

   (a) Four subjects stated that their strategy for resolving inconsistencies was to remove as few of the original axioms as possible. They tried to change the abstract ontologies as little as possible, but still make them valid. They assumed the ontologies were modelled purposefully by the modeller who tried to describe something, but mistakes were introduced carelessly, therefore, they had to remove something and retain as much information as they could. In this stimulus, the subjects would choose to change the classes at a lower level in the hierarchy in order to minimise the loss of information or the impact on the ontology. Moreover, more specific classes carry more information and they should be preserved. Thus, if a class has multiple parents which are disjoint with each other, then the subsumption relation with the least number of ancestors should be removed. For example, in Figure 5.2 (on the right-hand side), AX has multiple parents which are disjoint with each other, AX has four ancestors along axiom 1; it has two ancestors along axiom 5, therefore, axiom 5 is the candidate to be removed.

(b) Two subjects believed that it would be easier to introduce errors by mistake when modelling classes at a lower level in a hierarchy. This is because (1) some might forget the definitions of the upper classes; (2) the more superclasses a class has, the easier it is for the class to have a contradictory definition to its superclasses; (3) the more specific definitions of classes are unlikely to be correct, because few instances in the real world could fulfil such specific constraints. Therefore, for the example shown in Figure 5.2 (on the right-hand side), axiom 1 is the candidate to be removed.

(c) One subject believed that it was more destructive to change the class hierarchy, while removing disjointness or negated information had less impact on the ontology. Therefore, for the example shown in Figure 5.2 (on the right-hand side), axiom 7 (the disjointness) is the candidate to be removed.

(d) The remaining three subjects did not consider this factor.

5. Usage of classes – Six subjects did not consider this factor or believed it was not relevant. Only one subject believed classes highly referenced by other elements are more important, while one subject claimed that higher usage of classes could be over-modelled (e.g., it is unusual that a class contains numerous instances or has plenty of subclasses, compared to their classes), and more likely to be problematic. This heuristics is similar to the heuristic used in ontology evolution [142] which states that "if there are more than a dozen subconcepts for a concept, then an additional layer in the concept hierarchy may be necessary".

Besides considering the above factors, some subjects also introduced other factors to resolve the inconsistencies during the study.

- One of the subjects believed that expert ontology engineers were good at modelling ontologies; they usually did not introduce errors and did understand the whole ontology; whereas non-sophisticated users always had misunderstandings and needed help from tools/systems to resolve errors. Therefore, the subject tried to understand how and why ontology modellers built the ontology and introduced mistakes. The following factors were considered by this subject:

  1. The limitations of the ontology editors. Ontology editors usually display a concept hierarchy tree. An ontology may have concepts with multiple parents however, which cannot be directly represented in a tree. The typical solution for representing a concept with multiple parents is for the tree to display that concept multiple times, once for each path from the root to the concept. If, in addition, the concept exists at a deep level of a hierarchy, then it is very unwieldy to display it in the standard tree format. Hence it is difficult for users to browse a concept's multiple parents which might be disjoint with each other. Based on the subject's experience, he guessed that the problem only becomes obvious when the hierarchy length of an ontology is more than 5;

2. Ontology users' misunderstanding of OWL ontologies as discussed in [125]: misuse of existential and universal restrictions, confusion of 'some not' and 'not some', confusion between equivalence and subsumption relations etc.;
3. Reusing/integrating with other users' ontologies. It is common to build an ontology by adding more specific classes to the foundational ontologies. Note that foundational (or upper) ontologies usually provide a structure upon which ontologies for specific subject matters can be based. In this case, the more specific classes newly introduced in the ontology are likely to be incorrect if contradictions occur.

- One subject considered the pattern of the concept names, even though the stimuli in the experiment were abstract. He explained that classes with subsumption relations usually have similar patterns, e.g., matching prefix or suffix.
- One subject did not consider the five factors in the stimuli. She believed that if a class had contradictory information, the reasons would be (1) it kept too much information; the information of a class conceptually belonged to different concepts; or (2) there existed an exceptional situation in the class, and hence it should not be considered as one single class, but two different classes. In the study, her strategy was to split a class into two, so that two classes carried different information. For example, if the definition of a class CX is contradictory to that of its superclass, then the superclass should be split into two, so that one class fulfils the definition of class CX; another one fulfils the definition of other subclasses.
- We failed to get heuristics from one of the subjects. He claimed that the choice of axioms to be removed depends on the actual domain that was modelled. He explained that, without having access to the knowledge that is being represented, all solutions were equally valid; it was not possible to know how to change the ontologies.

	Subject 1	Subject 2	Subject 3	Subject 4
Size of axioms	Longer axioms are likely to be erroneous. Only the erroneous part of the longer axioms should be rewritten.	Longer axioms are likely to be erroneous; but also carry more information.	Removing shorter axioms is the simplest way to resolve problems.	Removing shorter axioms will cause less information to be lost.
Disjointness between sibling classes	(1) The disjoint classes usually have close common parents. (2) Ontology users usually do not understand disjoint axioms.	Sibling concepts should be disjoint with each other.	Disjointness usually exists between sibling concepts	Not considered.
Pattern of sibling classes	(1) Sibling classes are usually defined in a similar pattern (i.e., they usually have similar properties). (2) Ontology users usually misunderstand universal property restrictions.	Sibling concepts are usually at the same level of generality, i.e., participate in similar relationships.	Sibling concepts are usually defined in a similar pattern (i.e., they usually have similar properties).	It is usually erroneous to model a class that has a universal negated property filler (i.e., $\forall R.\neg D$), and the siblings of the class have the existential property restriction (i.e., $\exists R.D$).
Hierarchy level of classes	Subclass-of relations in a deep hierarchy are likely to be erroneous. This is because people would tend to forget the upper classes, and thus introduce errors.	Remove the class with the least subclasses and superclasses to minimise the loss of information.	Removing axioms in the lower levels of the hierarchy will cause less impact on an ontology.	The most specific axiom in a shallower hierarchy is chosen to be removed, in order to retain as much information as possible.
Usage of classes	Classes with higher usage should be removed, as they might be over-modelled.	Classes with higher usages may be more important, and hence should be retained.	Usage of classes are not relevant.	Not considered.

Table 5.1: Summary of the subjects' strategies for resolving inconsistencies (Continued in Table 5.2)

	Subject 5	Subject 6	Subject 7	Subject 8
Size of axioms	Longer axioms should be preserved, because they give more information.	Longer axioms are too complex and likely to be wrong.	Removing shorter axioms will cause less information to be lost.	Longer axioms are likely to be erroneous; but also carry more information.
Disjointness between sibling classes	Disjointness should be kept, as it is regarded as ‘strong’. ‘Strong’ means the modeller defines two classes to be completely different for strong reasons.	In this subject’s experience, the disjointness axiom is not likely to be asserted by mistake. In contrast, a subsumption relationship may often be asserted by mistake.	Disjointness usually exists between sibling concepts	(1) Sibling concepts should be disjoint with each other, (2) disjointness is more important than subsumption; removing disjointness has a bigger impact on the ontology.
Pattern of sibling classes	The universal restriction with a negated filler is too ‘strong’. Either none or all of the instances fulfill such a restriction. Modellers usually introduce this restriction after careful consideration.	The class with a universal restriction is likely to be erroneous, as its relation filler can be satisfied by an empty set; other siblings have an existential restriction.	Universal restrictions are ‘strong’, modellers usually introduce them purposefully, and therefore they are more likely to be correct.	The universal restrictions are too ‘absolute’ or ‘strong’, as either none or all instances fulfill such restriction; few instances in the real world would fulfill such a restriction.
Hierarchy level of classes	To retain more information, the most specific axiom in a shorter hierarchy is removed.	It is more likely to introduce errors when going deeper in organising the hierarchy, thus the classes at a lower level in the hierarchy are likely to be erroneous.	A change to the hierarchy structure is more ‘destructive’; disjointness axioms are less ‘important’.	Classes in the lower level of a hierarchy are likely to be correct, as their definitions are more specific. But the subject was not sure of the answer in this stimulus.
Usage of classes	Not considered.	Usage of classes is not relevant.	Not considered.	Usage of classes is not relevant.

Table 5.2: Summary of the subjects’ strategies for resolving inconsistencies (Continued)

5.2.6.2 General comments

The overall comment of the subjects was that they found it very hard to resolve the problems because the stimuli were abstract. If the ontology had a concrete domain, or if there were information about the process by which the ontology was constructed, and the stage at which inconsistencies were introduced, then it would be easier. For example, if a non-expert user extends an ontology downloaded from the Web, and unsatisfiability occurs, then the axioms introduced by the user are more likely to be erroneous. The subjects have different experience in modelling OWL ontologies; some have been working as ontology engineers in industrial ontology projects, some have been involved in Semantic Web projects, etc. Therefore, various heuristics for changing problematic axioms were obtained from their groups.

5.2.6.3 Conclusions

Heuristics from the subjects vary greatly; for example, some subjects emphasised the impact of removing axioms on the ontologies while others considered the common ontology modelling errors and patterns. Furthermore, our studies have revealed the following contradictory heuristics. Firstly, some subjects evaluated the likelihood of *correctness* of axioms, by making assumptions about the expertise of the original modellers, and on the other hand, some subjects evaluated the *importance* of axioms in ontologies, based on factors such as the loss of information due to removal.

	Correctness	**Importance**
Size of Axioms	Longer axioms are too complex and likely to be erroneous, and hence should be removed.	Longer axioms carry more information; their removal would then have a bigger impact on the ontology, and hence they are more important and should be retained.
Usage of classes	Higher usage of classes might be over-modelled (e.g., a class has numerous instances or subclasses).	Classes with higher usage are more important.
Length of a concept hierarchy	A class in a longer path in a hierarchy is likely to be erroneous, because it is easier for users to make mistakes at a deeper class hierarchy, and hence it should be removed.	A class in a deeper hierarchy is more specific and carries more information, it is thus more important in the ontology and hence should be preserved.

Furthermore, the correctness of axioms is dependent on *the level of expertise* of ontology modellers. It is common for a novice to make "common mistakes" (as enumerated by Rector et al. [125]) in OWL ontologies; an experienced ontology modeller is unlikely to introduce such mistakes.

	Non-Experts	**Experts**
Disjointness	Misunderstanding disjointness, such as complete partition.	Disjointness is usually asserted purposefully.
Universal restrictions	Misuse of universal restrictions instead of existential restriction as default quantifier.	Universal restrictions are usually asserted after careful consideration.

5.3 Related Work

5.3.1 Think-aloud Protocol

The empirical approach is similar to the one used by Korpi [97], and Alberdi et al. [1; 2]. There are main two difference between their studies and our studies. Firstly, Korpi and Alberdi et al. focused on inconsistencies in categorisation and taxonomy; while we are interested in logical inconsistencies in ontologies which are represented in Description Logics. Secondly, Korpi used instances of category consisting of words which represented commonly-occurring natural concepts (e.g., Cow). Alberdi et al. used real life botanical instances, so that subjects conducted a task that was closely related to their real life scientific activities. In contrast, when we used both concrete (meaningful English terms) and abstract (anonymous terms) stimuli in the first pilot and second studies, we failed to draw any generic heuristics from the subjects doing concrete examples as they drew on their personal knowledge of the domain. In some cases with concrete stimuli, some subjects still had to change certain parts of the ontologies even that were logically consistent. On the other hand, with abstract stimuli, the subjects considered the impact of changes on ontologies and the structure of the ontologies. We believe this is a realistic way to simulate in debugging ontology situations. This is because the subjects could only apply their extensive OWL ontology and DLs knowledge to solve problems. That means, they had to rely on their experience of working in Semantic Web projects, or common good practices of modelling ontologies, rather than commonsense knowledge. Therefore, the acquired heuristics could be applied in general ontologies.

5.3.2 Ontology Repairing

The most relevant related work providing support for resolving errors in OWL ontologies is SWOOP [91]. The authors propose various strategies to rank erroneous axioms to be removed from the MUPSs (Minimal unsatisfiability-preserving sub-TBoxes) [131] in order to fix the unsatisfiable concepts (described in Section 3.4.3). We now compare their strategies with the results of our studies:

1. Arity of axioms – in SWOOP, an axiom with a higher arity value is preferred to be removed. In our studies, we only provided stimuli with a single inconsistency, therefore, we cannot

conclude if the arity of axioms is a factor for subjects to resolve errors.

2. Test cases – the authors of SWOOP mention allowing the user to specify their own test cases manually. However, this functionality is not yet implemented in SWOOP. We did not investigate test cases in our studies.

3. Provenance information about axioms (author, source reliability, timestamp etc.) – The SWOOP authors mention that all changes made to an OWL ontology were kept, the change-log of ontologies can be shared, and that axioms added by a user with a high precedence level will be given high importance. However, the details are sketchy in their paper. In our studies, we did not include provenance information in the stimuli presented. However, we found that provenance information could be useful in resolving such tasks in the studies. This is because some subjects considered whether the ontology was constructed by a non-expert user, or the ontology was downloaded from the third party and further extended by the user.

4. Relevance to the ontology in terms of its usage – The authors of SWOOP suggest that the relevance of an element to the ontology depends on the usage of the element in its ontology. That means, if a concept is highly referenced by other elements in an ontology, then it is more relevant to the ontology, and hence more important, and so should be preserved. In our studies, only two subjects considered the usage of classes in the ontologies. One of them suggested that the class which is the least referenced is the prime candidate for removal, but another one claimed that higher usage of classes would should be removed, as they might be over-modelled (For example, a class is over-modelled if it contains too many instances or subclasses).

5. Impact on an ontology when removing axioms – SWOOP ranks axioms based on the number of entailments which would be broken if a particular axiom is removed. This implies that an axiom whose removal breaks more relationships is more important in the ontology. This factor is in agreement with the results of our studies, in which the axioms causing more broken relationships are more important and hence should be preserved.

Overall the comparison with SWOOP shows that some of the SWOOP heuristics are inline with the intuitions of our subjects, some were not considered by our subjects, but most significantly, our studies have uncovered several other heuristics which SWOOP has not considered; for example, the size of axioms, common ontology modelling errors, knowledge of ontology structures, and limitations of ontology editors.

5.4 Conclusion

In this chapter, we discovered a number of heuristics used by ontology engineers to resolve inconsistencies in ontologies. We discovered that it was necessary to work with abstract ontology examples in order to gather generic heuristics. The heuristics gathered include the impact of removal on the ontologies, knowledge of ontology structures, common OWL modelling errors, limitation of ontology editors etc. Our expectation was corroborated by the fact that some of the heuristics we ‘rediscovered’ are in fact already in use in the SWOOP tool, whose usefulness has been evaluated with concrete examples [91]. Interestingly, our studies also revealed some contradictory heuristics. A solution to these conflicting heuristics could be achieved by making more information available. If ontology projects could record information about the process by which the ontologies are constructed, the stage at which inconsistencies are introduced, and the experience level of ontology modellers, then debugging tools could use this information provide suitable heuristics. We believe that knowledge of these heuristics could be invaluable to those who build ontology debugging tools to help users resolve inconsistencies.

With a variety of heuristics, the next step is to formalise the heuristics, and then incorporate them into an ontology debugging tool. We are making a *major* assumption, namely that the heuristics which have been obtained in the studies from subjects working with abstract stimuli are in fact applicable when subjects work with concrete stimuli. To evaluate the practical usefulness of these heuristics, we have conducted a usability study with a number of unsatisfiable ontologies and subjects, we report the results in Chapter 8.

Chapter 6

A Heuristic-based Service for Selecting Problematic Axioms

This chapter describes a heuristic-based service to support a user to select problematic axioms to resolve unsatisfiable concepts. The notion of *confidence* is introduced. If an axiom is assigned a high confidence value we recommend that the axiom should be preserved. If an axiom is assigned a low confidence value this means that we recommend that the axiom should be removed or modified. Confidence values are assigned by a set of heuristics. Some of these heuristics were acquired from our empirical studies reported in Chapter 5 and some are adapted from the literature. Three types of heuristics are distinguished: (i) impact of removal on the ontology, (ii) knowledge of ontology structure, and (iii) a linguistic heuristic.

This chapter is organised as follows: Section 6.1 presents the motivation for our approach to selecting axioms. Section 6.2 describes existing approaches. In Section 6.3, we formalise heuristics to rank potentially problematic axioms in ontologies. We discuss the fine-grained approach to ranking parts of axioms in Section 6.4. Our approach is compared with existing related work in Section 6.5. Finally, conclusion is presented in Section 6.6.

6.1 Motivation

As presented in Chapter 1, existing approaches [131; 154; 112] typically identify problematic axioms for unsatisfiable concepts in an OWL ontology, and then leave it up to the user to resolve the detected errors. We claim that methods are needed to rank potentially problematic axioms to give the user guidance on which axioms should be removed or modified. We now present two typical scenarios of unsatisfiable ontologies to illustrate the need for such methods.

6.1.1 Common Errors in OWL Ontologies

Firstly, suppose that a non-expert user either reuses an ontology downloaded from the Web or creates it from scratch. Such an ontology might contain unsatisfiable concepts; a DL reasoner can check the ontology and find the unsatisfiable concepts. Common mistakes include the misuse of universal restrictions as in, and confusion the difference between non-primitive (i.e., equivalence) and primitive classes (i.e., subsumption relations) [125; 153]. For example, given an axiom $\mathsf{Taxi\_Driver} \doteq \ \forall\, \mathsf{drives.Taxi}$, it is obvious for the user that everyone who drives a taxi must be a taxi driver, but by using the universal restriction and the equivalence relation it is implicit that any instance in the ontology which does not drive will also be classified as a taxi driver. Non-experts are typically unaware of this implicit information. If the domain of drives is Driver, and Taxi_Driver is a subclass of Driver, then any concept which is disjoint with Driver will be unsatisfiable (because Driver is implicitly equivalent to owl:Thing).

Example 29 *Given that an ontology containing the following axioms is downloaded by a user,*
1. Taxi_Driver $\sqsubseteq$ Driver
2. domain(drives) = Driver
3. Cat $\sqsubseteq$ $\neg$ Driver
4. Bus_Driver $\sqsubseteq$ $\exists$ drives.Bus $\sqcap$ Driver
5. Van_Driver $\sqsubseteq$ $\exists$ drives.Van $\sqcap$ Driver
6. Lorry_Driver $\sqsubseteq$ $\exists$ drives.Lorry $\sqcap$ Driver

If the user extends this ontology by adding a new axiom:
7. Taxi_Driver $\doteq$ $\forall$ drives.Taxi
then Cat *will become unsatisfiable. A DL reasoner identifies axioms 1, 2, 3 and 7 as responsible for the unsatisfiability. However, the user might find it difficult to determine which axiom(s) are really problematic.*

In this example, we notice that axiom 7 is an equivalence relation, and contains a universal restriction; it matches the pattern of the common errors described before. However, it is not convincing to say that any equivalence axiom or any universal restriction is incorrect. But with the sibling pattern heuristic mentioned in Chapter 5, which states that 'sibling concepts should have similar patterns', we analyse the structure of sibling concepts, and discover that Taxi_Driver is modelled differently compared with its siblings. Therefore, axiom 7 is most likely to be erroneous.

6.1.2 Merging Ontologies

Secondly, online ontologies are created for different purposes, and in different contexts, hence merging these ontologies might result in inconsistencies. For example, when the user merges two well-established ontologies, such as the SUMO.owl[1] and CYC.owl[2] upper ontologies, thus merged ontology might contain a large number of unsatisfiable concepts (as shown in [130]) because the shared concepts (e.g., Entity exists in both ontologies) are modelled to have different meanings. It is difficult for the naive user of the two ontologies to resolve the errors due to being unfamiliar with the different domain terminologies. For another example, in [113] , an inconsistency results from merging two ontologies, in which a shared concept "bass" is defined to have two meanings. One ontology has the meaning of "bass as a singer"; the other ontology has the meaning of "a bass that lives in a certain river". It is common for inconsistencies in ontologies to be caused by concepts with multiple definitions [113; 84].

[1]http://ontology.teknowledge.com
[2]http://www.opencyc.org

6.2 Existing Approaches

In this section, we present the existing approaches for dealing with unsatisfiable concepts. Schlobach [131] first presented an algorithm to pinpoint the problematic axioms causing the unsatisfiability of concepts. Kalyanpur et al. [91] then proposed strategies for ranking the problematic axioms in order of importance.

6.2.1 Pinpointing Axioms

Schlobach [131] proposes an algorithm to pinpoint logical contradictions in a description logic knowledge base. By 'debugging' the author means the identification and elimination of modelling errors when detecting logical contradictions in a knowledge base. In order to identify the precise position of errors within a TBox, minimal unsatisfiability-preserving sub-TBoxes (abbreviated MUPS) and minimal incoherence-preserving sub-TBoxes (abbreviated MIPS) are introduced. The method of calculating MUPSs is described in Chapter 4. In simple terms, a MUPS of a concept is a minimal set of axioms causing the concept to be unsatisfiable.

Definition 1 (MUPS) *Let C be an unsatisfiable concept in a TBox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a MUPS of C if C is unsatisfiable in $\mathcal{T}'$, and C is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$.*

Example 30 *Given an ontology $\mathcal{O}$ with the following axioms:*
α_1. $A \doteq B \sqcap E$
α_2. $B \doteq \neg C \sqcap E$
α_3. $E \doteq C \sqcap F$
α_4. $F \doteq \neg C \sqcap \forall R.C$
α_5. $G \sqsubseteq F$
α_6. $H \sqsubseteq \neg C \sqcap \forall R.C$

Concept E is unsatisfiable because it is a subclass of both C and $\neg C$; concept B is unsatisfiable because it is a subclass of both $\neg C$ and C, and it is also a subclass of unsatisfiable E; concept A is unsatisfiable because it is a subclass of two unsatisfiable concepts, B and E. Their MUPSs are:
$MUPS(A) = \{\{\alpha_1, \alpha_2, \alpha_3\}, \{\alpha_1, \alpha_3, \alpha_4\}\}$
$MUPS(B) = \{\{\alpha_2, \alpha_3\}, \{\alpha_2, \alpha_3, \alpha_4\}\}$
$MUPS(E) = \{\{\alpha_3, \alpha_4\}\}$

Minimal incoherence-preserving sub-TBoxes (MIPS) are the smallest subsets of an original TBox preserving unsatisfiability of at least one atomic concept. In brief, a MIPS of an ontology is the minimal set of axioms causing the ontology to be unsatisfiable (i.e., incoherent).

Definition 2 (MIPS) *Let $\mathcal{T}$ be an incoherent TBox. A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a minimal incoherence-preserving sub-TBox (MIPS) of $\mathcal{T}$ if $\mathcal{T}'$ is incoherent, and every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent.*

Continuing with Example 30, removing axiom α_1 can resolve the unsatisfiability of A; removing axiom 2 can resolve B; removing axiom α_3 or α_4 can resolve E. There are many combinations of removing axioms to resolve these unsatisfiable concepts. Actually, there are two true

causes of the unsatisfiability: E is a subclass of both C and $\neg C$ (due to axioms $\{\alpha_3,\alpha_4\}$); B is a subclass of both $\neg C$ and C (due to axioms $\{\alpha_2,\alpha_3\}$). If the two contradictions are resolved, then A becomes satisfiable. That means, we only have to remove one axiom from the sets $\{\alpha_3,\alpha_4\}$ and $\{\alpha_2,\alpha_3\}$, which are the smallest subsets of the original ontology, preserving unsatisfiability of at least one atomic concept. For this ontology, we obtain the MIPS:
$MIPS(\mathcal{O}) = \{\{\alpha_2, \alpha_3\}, \{\alpha_3, \alpha_4\}\}$.

From this MIPS, we can see that removing axiom α_3 can resolve two unsatisfiabilities; removing axiom α_2 can resolve one unsatisfiability. The number of occurrences of an axiom in the sets of the MIPS is denoted as *arity*, i.e., $arity(\alpha_3) = 2$; $arity(\alpha_2) = 1$. The higher the arity value of an axiom, the more likely it is that this axiom will be the cause of contradictions.

6.2.2 Ranking Axioms

Kalyanpur et al. [91] make use of arity to rank problematic axioms in order of *importance*. The aim is to remove the lowest ranked axioms from the ontology, and preserve the highest rank ones. In their approach, the higher the arity value of an axiom, the lower the rank assigned to the axiom.

Further they consider the impact of axiom removal on the ontology. When removing an axiom, they check which subsumption relations (between atomic concepts) and/or instantiation (of atomic concepts) would break. Axioms to be removed are ranked based on the number of entailments they break (the higher the rank, the fewer the entailments broken). This strategy is consistent with the humans' heuristic obtained in Chapter 5, in which subjects' strategies for resolving inconsistencies are to remove some axiom yet retain as much information as possible.

Finally, the relevance to the ontology in terms of its usage is also taken into account when ranking is performed. If the entities (atomic concepts, properties, or individuals) in the axiom are referred to often in the remaining axioms or assertions of the ontology, then the entities are in some sense, central to the overall theme of the ontology, and hence removing axioms related to these entities may be undesired. For example, if a concept is heavily instantiated, then changing the axiom definitions of that concept is not desirable.

6.2.3 Limitations of Existing Work

It is important to minimise the impact of removal of axioms on the ontology. Intuitively, an axiom whose removal causes more information to be lost is regarded as more important, and should be preserved. We reuse the example in Chapter 1 to show the limitation of the existing work.

Example 31 *The ontology contains the following axioms:*

(1) lions $\sqsubseteq \exists$ eat.(cows $\sqcap$ sheep)	[lions eat something which is both cow and sheep]*;*
(2) tigers $\sqsubseteq \exists$ eat.(cows $\sqcap$ sheep)	[tigers eat something which is both cow and sheep]*;*
(3) cows $\sqsubseteq \neg$ sheep	[cows and sheep are disjoint with each other]

Axioms (1) and (3) cause lions *to be unsatisfiable; axioms (2) and (3) cause* tigers *to be unsatisfiable. In this case, axiom (3) causes two unsatisfiable classes; its removal can resolve two unsatisfiabilities. Also removing axiom (3) is a change which preserves more of the information in the ontology, as otherwise two axioms would have to be removed (axioms (1) and (2)). By removing (3) we can retain the information of* lions *and* tigers. SWOOP *uses both of these heuristics,* *i.e., number of unsatisfiabilities caused by the axiom, and information lost by a modification. As a result,* SWOOP *assigns axiom (3) less importance, and suggests the user removes or modifies axiom (3).*

The above example shows SWOOP's recommendation, however, it does not necessarily follow that such suggestions are the best modifications for all ontologies. A human ontology engineer may well come up with a better solution in particular cases. For example, a human may believe that sibling classes are usually disjoint with each other. That means that a disjointness between siblings should be kept, even if that causes many concepts to be unsatisfiable (i.e., it has a high arity value and hence higher importance). This heuristic has an opposite effect to the factor of information lost.

As we mentioned in Section 1.3, it is challenging for software tools to determine the optimal way to resolve such errors (semi)-automatically in every ontology. Hence many tools simply leave it up to the user to determine the optimal way to resolve such errors. However, our hypothesis is that it is useful to formalise the heuristics humans use to solve these kinds of problems, and to incorporate these heuristics in a tool. In this chapter, we will combine humans' heuristics obtained from the empirical studies in Chapter 5 with existing heuristics (e.g., arity and impact of removal) to deal with the inconsistencies in ontologies.

6.3 Confidence of Axioms

As described above, the MUPS (which can be calculated by the tableau algorithm described in Section 4.2) shows us a set of axioms from which one must be removed to resolve an unsatisfiability; however we need to identify which axiom should be removed or modified. To do this we assign *confidence* values to axioms. If an axiom is assigned a high confidence value this means that we recommend that the axiom should be preserved. If an axiom is assigned a low confidence value this means that we recommend that the axiom should be removed or modified. Confidence values are assigned by a set of heuristics; an example is the "impact" heuristic which assigns high confidence to axioms whose removal would impact many of the named concepts in the ontology (the aim is to minimise the loss of information from the ontology).

6.3.1 Definition of the Confidence of Axioms

The confidence function below provides a ranking over the axioms, allowing us to indicate axioms recommended for removal or preservation. As we have to compare the relative confidence values of axioms, the heuristic formulas below transform the computed values into the range $[-1, 1]$ (normalisation). Axioms are initially presumed to have 0 confidence; axioms with values close to $+1$ are strongly recommended for preservation, while those with values close to -1 are strongly recommended for removal. Axioms for which a confidence value is not determined for a particular heuristic are assigned the default confidence value 0.

Definition 3 (Confidence) *Let α be an axiom in an ontology $\mathcal{O}$, the confidence of α is a function*

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

We have acquired a variety of heuristics from our empirical studies in Chapter 5, the next step is to formalise the heuristics which allow us to evaluate the confidence of axioms. We combine our heuristics with those already incorporated in existing debugging tools (i.e., arity and impact of removal) and divide them into the following categories:

1. Impact of removal on the ontology – two adapted heuristics are presented to analyse the loss of entailments due to the removal of axioms; these heuristics are inline with the subjects' heuristics in Chapter 5, which was to remove as little of the original axioms as possible, in order to reduce the impact on the ontology.

2. Knowledge of ontology structure – five heuristics acquired from the subjects' heuristics in Chapter 5 are presented:

   (a) disjointness between siblings,

   (b) disjointness between non-siblings,

   (c) sibling patterns,

   (d) length of paths in a concept hierarchy, and

   (e) depth of concepts in a concept hierarchy

3. Linguistic heuristic – this heuristic is also acquired from our subjects, it is presented to analyse the similarities of the labels of the concepts

4. Other heuristics including the size of axioms, lexical-syntactic patterns and usage are presented. These are not evaluated by the usability study in Section 8.2, as there is not a sufficient number of realistic unsatisfiable ontologies on which they can be tested. They are presented because we believe that they constitute a potentially interesting direction of future work.

Each of these is addressed in the sections below.

6.3.2 Impact of Removal of Axioms

From the empirical studies in Chapter 5, the subjects' heuristic was to remove as little of the original axioms as possible, in order to reduce the impact on the ontology. For example, the classes at the lowest level in a hierarchy are usually chosen for removal, and so none of its subclasses and superclasses will be affected. This heuristic is also in agreement with SWOOP's strategies and the *Principle of Minimal Change* [95]. This principle is an important issue which is often considered in belief revision algorithms [3]. The principle states that the changed knowledge base (KB) should be as "close" as possible to the original one. That means one should choose modifications which have minimal impact on the existing ontology [63]. Currently, there is no consensus as to how this principle should be formally expressed [40]. Hasse et al. [63] claim that the definition of minimal impact may depend on the particular user requirements; the authors simply count information loss as being equal to the number of removed axioms, in order to achieve a maximally consistent sub-ontology.

The above intuition can be captured by the notion of *arity* [131]. The arity of an axiom α is the number of unsatisfiable concepts that have at least one MUPS containing α. Axioms occurring in the MUPSs of many different concepts cause many contradictions, so their confidence value is close to -1.

Definition 4 (Arity of Axioms) *Given an axiom α involved in any unsatisfiable concept in an ontology, its confidence is defined as*

$$conf_{arity}(\alpha) = \frac{1 - arity(\alpha)}{arity(\alpha)}$$

Besides counting the number of axioms removed from the ontology, in our approach, we also consider the impact of removal on the named concepts in the ontology. Given a modified ontology and its original version, the closeness of ontologies could be defined in terms of (1) the number of satisfiable named concepts that will be affected, and (2) the number of the lost entailments of named concepts in the ontology due to the removal of an axiom.

(1) Number of affected named concepts

The following example explains why the number of affected named concepts is necessary, in addition to the number of lost entailments.

Example 32 *Given a terminology with three axioms: $\alpha_1 : C \sqsubseteq C_1$, $\alpha_2 : C_1 \sqsubseteq C_2$, $\alpha_3 : C_2 \sqsubseteq C_3$. If $\alpha_3 : C_2 \sqsubseteq C_3$ is removed, then three entailments will be lost (i.e., $C_2 \sqsubseteq C_3$, $C_1 \sqsubseteq C_3$, and $C \sqsubseteq C_3$) and three concepts will lose their superconcepts. If $\alpha_1 : C \sqsubseteq C_1$ is removed, then three entailments will also be lost (i.e., $C \sqsubseteq C_1$, $C \sqsubseteq C_2$, and $C \sqsubseteq C_3$), but only C will lose its superconcepts. Therefore, we say removing α_3 has a bigger impact on the ontology, and hence has a higher confidence value.*

(2) Number of lost entailments of named concepts

We reproduce Example 30 as follows, to explain what we mean by the impact of removal axioms from an ontology.

α_1. $A \doteq B \sqcap E$

α_2. $B \doteq \neg C \sqcap E$

α_3. $E \doteq C \sqcap F$

α_4. $F \doteq \neg C \sqcap \forall R.C$

α_5. $G \sqsubseteq F$

α_6. $H \sqsubseteq \neg C \sqcap \forall R.C$

When an axiom involved in the unsatisfiability of a concept is changed, we calculate the impact of removal on the ontology in three ways:

1. *the named concepts involved in the unsatisfiability*: In this example, E is unsatisfiable due to axioms α_3 and α_4, if α_4 is removed, then the indirect assertions $E \sqsubseteq \forall R.C$, $B \sqsubseteq \forall R.C$, and $A \sqsubseteq \forall R.C$ will be lost, because the part '$\forall R.C$' in α_4 is not responsible for the unsatisfiability of E, B, and A;

2. *the satisfiable named concepts irrelevant to the unsatisfiability*: Other satisfiable named concepts irrelevant to the satisfiability might lose entailments introduced by the axiom to

be changed. The entailments we consider in this case are indirectly asserted in the ontology before the change. If the user removes the problematic part '$\neg C$' in α_4, then all the subconcepts of F will be affected. The indirect assertion $G \sqsubseteq \neg C$ will be lost;

3. *the classification of the named concepts of the ontology*: The satisfiable named concepts might lose implicit subsumption relations due to the change of axioms. In the example, H is an implicit subclass of F due to axioms α_4, α_6. The change of α_4 might affect the implicit subsumption between H and F.

The methods of calculating the lost entailments when (parts of) axioms are removed are described in Chapter 7. These lost entailments may or may not constitute entailments which are missing from the changed ontology. To check if these entailments which are removed (due to the removal of axioms) are really lost entailments, we have to check if the changed ontology still satisfies these entailments. If it does not, then we can say these entailments are lost.

Definition 5 (Impact of Removal) *Given an axiom α to be removed from the ontology $\mathcal{O}$, and the changed ontology $\mathcal{O}' := \mathcal{O} \setminus \{\alpha\}$,*

- $\Omega = \{\beta | \mathcal{O} \models \beta$ *and* $\mathcal{O}' \not\models \beta\}$, *where β is of the form* $\mathsf{CN} \sqsubseteq D$ *(*CN *is a named concept, D is any concept description). The number of the lost entailments is $|\Omega|$.*
- $\mathcal{AC} = \{\mathsf{CN} | \mathsf{CN}$ *is the left-hand side of all* $\beta \in \Omega\}$. *The number of affected concepts is $|\mathcal{AC}|$.*

The confidence of α is defined as

$$conf_{impact}(\alpha) = \frac{|\Omega| + |\mathcal{AC}|}{|\Omega| + |\mathcal{AC}| + 1}$$

The result will be within [0,1). If the removal of an axiom α will not result in the loss of any entailment, and no other satisfiable named concepts are affected (i.e., $|\Omega| = |\mathcal{AC}| = 0$, as α is redundant), then the confidence value is 0. When the number of lost entailments or affected concepts increases, the confidence value becomes close to 1. For example, if a disjoint axiom exists between two concepts C and D, which have many subconcepts, then removing this axiom removes the disjointness between all the subconcepts of C and D and hence its confidence value is close to 1.

6.3.3 Knowledge of Ontology Structure

The empirical studies revealed that the subjects exploit knowledge of the ontology structure to evaluate the correctness or importance of axioms in ontologies. For example in their view, a longer path in a concept hierarchy is likely to be erroneous compared to the shorter ones. In this section, we formalise these heuristics and name them as *structural heuristics*, which are classified into five types:

1. Disjointness amongst Siblings – Disjointness axioms usually exist between sibling concepts.
2. Disjointness amongst Non-Siblings – Disjointness axioms existing between non-siblings are likely to be incorrect.

3. Pattern of sibling concepts – The siblings in the concept hierarchy should be at the same level of generality [115], they are usually defined in a similar manner.
4. The length of paths in a concept hierarchy – A longer path in a concept hierarchy is likely to be erroneous.
5. The depth of concepts in a hierarchy – Concepts at the upper level of a concept hierarchy are likely to be more important.

Each of them is explained in more detail below.

6.3.4 Disjointness amongst Siblings

In general the subjects in the empirical studies in Chapter 5 believed that sibling classes are usually disjoint with each other. This is inline with the *Strong Disjointness Assumption* (SDA) [32] which studies that: in a well-modelled ontology the direct siblings, i.e., children of a common parent in the subsumption hierarchy should be disjoint. Also, Gómez-Pérez et al. [54] and Rector et al. [125] state that one of the common errors in building ontologies in OWL has been to omit disjoint axioms. Therefore, if disjoint axioms which cause unsatisfiability exist among sibling concepts, then we assume that the ontology modeller is really sure that the concepts are completely different when asserting such axioms. In this case, these axioms are likely to be correct, and hence we increase their confidence values. If there exists a complete partition [54] among siblings, we suspect that the ontology modeller not only understands the use of disjoint axioms, but strongly believes that the instances of the sibling concepts cannot overlap and are mutually exclusive. Therefore, we assign confidence values of 1 to disjoint axioms in the complete partition. In general, we have more confidence in the disjoint axioms if more sibling concepts are disjoint with each other. We capture this by the following definition:

Definition 6 (Disjointness amongst Siblings) *Given n sibling concepts in an ontology, and m disjointness axioms existing between them, and α is one of the disjoint axioms, then*

$$conf_{disjSib}(\alpha) = \frac{m}{n(n-1)/2},\ where\ n \neq 0\ and\ n \neq 1$$

The result of the above confidence calculation will be in the range $[0, 1]$. If α is the only disjoint axiom between the n sibling concepts, then its confidence will approach 0 as n increases, and hence this axiom is a likely candidate for removal. If there are n sibling concepts, and a complete partition amongst all the siblings exists, that means there are $n(n-1)/2$ disjoint axioms, then the confidence value of those axioms will be 1, and the axioms should be preserved.

6.3.5 Disjointness amongst Non-Siblings

Moreover, if the disjoint axioms which cause the unsatisfiability exist amongst non-sibling concepts, our subjects believed that the ontology modeller might have misunderstood the definition of concepts or disjointness, and made a mistake. In this case, these axioms are likely to be incorrect, and hence we decrease their confidence values close to -1. This is captured via the following definition:

Definition 7 (Disjointness amongst Non-Siblings) *Given a disjoint axiom α between named concepts A and B, where C is the nearest common ancestor concept, n is the number of intermediate named concepts between A and C in the named concept hierarchy; m is the number of intermediate named concepts between B and C in the concept hierarchy, then*

$$conf_{disjNS}(\alpha) = \frac{-|\frac{m+n-2}{m+n}| - |\frac{m-n}{m+n}|}{2}$$

The result of the above confidence calculation will be in the range $[-1, 0]$. The first part $-|\frac{m+n-2}{m+n}|$ measures the distance of the concepts from their common ancestor. If their distance from the common ancestor is longer, the axiom is likely to be incorrect. That is, the confidence value approaches -1 when n or m increase. If n and m are both equal to 1, then the concepts are siblings, and we do not decrease its confidence value. The second part $-|\frac{m-n}{m+n}|$ measures the level difference between n and m in the hierarchy. If the difference $|n - m|$ between their hierarchy levels is larger, then the axiom is likely to be incorrect, and then the confidence value is close to -1. If the concepts are at the same level, i.e., $m = n$, the confidence value is not decreased.

6.3.6 Pattern of Sibling Concepts

Some subjects in the studies believed that sibling concepts in a hierarchy are usually defined in a similar pattern, this means that they have similar restrictions and generality. This is consistent with the guidelines for creating ontologies produced by Noy and McGuinness [115], which says 'all the siblings in the hierarchy (except for the ones at the root) must be at the same level of generality'.

For the example used in the empirical studies in Chapter 5 (reproduced in Figure 6.1), AX is unsatisfiable because at least an instance of AX has a has-R relation with D2, but all of its instances do not have a has-R relation with D2.

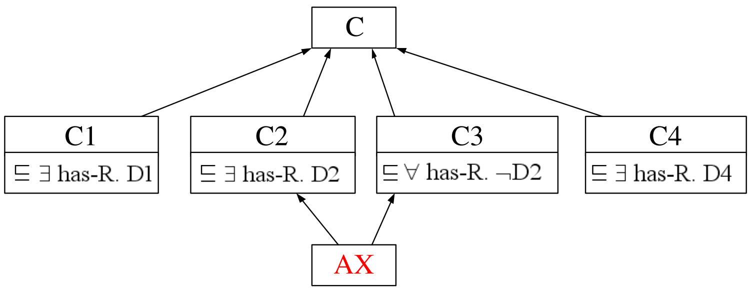


Figure 6.1: An example illustrates the pattern of sibling concepts

Some subjects believed that C3 $\sqsubseteq$ $\forall$ has-R.$\neg$ D2 was an error, because the class C3 was defined in a different manner to its siblings. The other three sibling classes had an existential restriction $\exists$has-R; the subjects expected that C3 should have an existential restriction $\exists$ has-R.D3 too, then all siblings should have the same pattern, and D3 could be equivalent to $\neg$D2. Therefore they would remove C3$\sqsubseteq$ $\forall$ has-R.$\neg$D2, and add C3$\sqsubseteq$ $\exists$has-R.D3. One of the subjects used the following concrete example to explain. If Bus_Driver is defined to drive at least one Bus, then in the majority of cases, the ontology modeller would not define a Car_Driver as someone who never drives a Bus.

In addition, some subjects claimed that C3 $\sqsubseteq$ $\forall$ has-R.$\neg$ D2 was modelled incorrectly, and it might have been introduced by a non-expert user, and non-expert users are usually unclear that universal restrictions can be "trivially satisfied". Therefore, in the following, we propose a heuristic which combines the sibling patterns and common error patterns to evaluate the confidence of axioms.

Rector et al. [125] enumerate a number of common error patterns made by non-expert users in modelling OWL ontologies which are shown in Table 6.1. The 'Asserted' column shows the pattern of axioms usually asserted by the user wrongly, actually, the pattern of axioms in the 'Intended' column should be the intended meaning.

	Asserted	**Intended**
Confusion on the difference between the linguistic and logical usage of 'and' and 'or'	$C_1 \sqcap C_2$	$C_1 \sqcup C_2$
Use of universal restrictions rather than existential restrictions as the default quantifier	$\forall R.C$	$\exists R.C$
Misuse of equivalence relations instead of subsumption relations	$A \doteq C$	$A \sqsubseteq C$
Confusion on the representation of 'some not' and 'not some'	$\exists R.\neg C$ $\neg\exists R.C$	$\neg\exists R.C$ $\exists R.\neg C$

Table 6.1: Common Error Patterns

If an axiom is of the form CN $\sqsubseteq D$ or CN $\doteq D$, and (parts of) the axiom are identified to be responsible for a concept's unsatisfiability, then we compare the (parts of) axiom with the common error patterns, as well as with CN's siblings. For example, given an axiom $A \sqsubseteq \forall R.C \sqcap \exists R^*.(C^* \sqcap D^*) \sqcap B$, where concept and role names tagged with (*) indicate that they are responsible for the unsatisfiability. Obviously, only the parts tagged with (*) should be evaluated by the heuristics with a confidence value by comparing with CN's siblings.

We further explain the purpose of comparing with concepts' siblings as follows. If the pattern of an axiom matches the common error patterns, then we assign this axiom with a negative confidence value to indicate its incorrectness. It is difficult to determine the degrees of incorrectness in this case. Also, it easily happens that a set of problematic axioms all match with the common error patterns, we cannot simply assign these axioms with the same negative confidence value. Therefore, we propose a mechanism to assign confidence with different degrees by comparing with the patterns of concepts' siblings as follows.

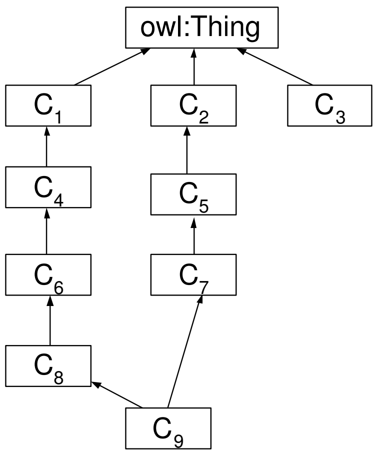
By analysing the sibling patterns and common error patterns, we can evaluate the correctness of an axiom, α, of the form CN $\sqsubseteq D$ or CN $\doteq D$ (where CN is a named concept, D is any concept description), with two conditions: (1) if the pattern of the (parts of) axiom is matched with the form of asserted one (the 'Asserted' column in Table 6.1); and (2) if the patterns of CN's siblings match the form of the intended one (the 'Intended' column in Table 6.1), then we suspect this axiom is likely to be erroneous, and assign it with a negative confidence value. Additionally, if the number of CN's siblings which match the pattern of the intended one increases, then the confidence of the axiom is decreased. In brief, if CN has no sibling concept, then its confidence is the default value, i.e., zero; however, if CN has many siblings whose patterns are the form of the intended one, then α is more likely to be erroneous, and hence has a lower confidence value. The result of the confidence calculation given below will be in the range $(-1, 0]$.

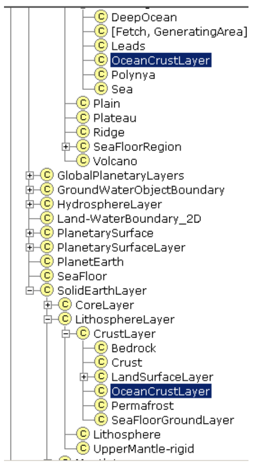
Definition 8 (Pattern of Siblings) *Given an axiom α whose left-hand side is a named concept* CN*, if (parts of) α match a pattern of the asserted form, and there are n sibling concepts whose assertions match a pattern of the intended form, then the confidence of α is defined as*

$$conf_{sib}(\alpha) = \frac{-n}{n+1}$$

6.3.7 Length of Paths in a Concept Hierarchy

In an ontology's conceptual model, a concept hierarchy is typically expressed in a directed acyclic graph (DAG) showed in the left-hand side of Figure 6.2. Each node represents a concept and each directed arc represents a subclass relation to present the hierarchical structure between concepts in ontologies. In DAG, a path is a distinct trace that can be taken from a specific particular concept to the most general concept in the ontology, which is the concept without any parent or superclass (e.g., C_9-C_8-C_6-C_4-C_1-owl:Thing in the left-hand side of Figure 6.2). The length of a path of a particular concept indicates the semantic distance between the concept and the most general concept, so the path length is defined as the sum of subclass relations on a path. If there is more than one path from a concept to the top concept, then the longest path is always assigned to it. The length of the path of concept C_9 is 5 in Figure 6.2.





```
DeepOcean
[Fetch, GeneratingArea]
Leads
OceanCrustLayer
Polynya
Sea
Plain
Plateau
Ridge
SeaFloorRegion
Volcano
GlobalPlanetaryLayers
GroundWaterObjectBoundary
HydrosphereLayer
Land-WaterBoundary_2D
PlanetarySurface
PlanetarySurfaceLayer
PlanetEarth
SeaFloor
SolidEarthLayer
  CoreLayer
  LithosphereLayer
    CrustLayer
      Bedrock
      Crust
      LandSurfaceLayer
      OceanCrustLayer
      Permafrost
      SeaFloorGroundLayer
    Lithosphere
    UpperMantle-rigid
```

Figure 6.2: Left-hand side: Ontology represented in DAG. Right-hand side: OceanCrustLayer in Sweet-JPL.owl ontology is unsatisfiable and has multiple parents, the length of its paths is 7.

One of the common causes of unsatisfiable concepts is that a concept has multiple parents which are disjoint with each other. Some subjects in our studies remarked that some ontology editors display a concept hierarchy tree, but an ontology may have concepts with multiple parents, which cannot be directly represented in a tree. The typical solution for representing a concept with multiple parents is for the tree to display that concept multiple times, once for each path from the root to the concept. If, in addition, the concept exists at a deep level of a hierarchy, then it is very unwieldy to display it in the standard tree format. Hence it is difficult for users to browse a concept's multiple parents which might be disjoint with each other. In the right-hand side of Figure 6.2, OceanCrustLayer is unsatisfiable because its multiple parents are disjoint with each other. By browsing the hierarchy tree of the editor, it is not easy for the user to notice that its multiple parents are actually disjoint with each other. In this case, we believe that a longer path in the hierarchy which is involved in the unsatisfiability is likely to be erroneous compared to the shorter ones. We therefore decrease the confidence values of axioms in a path in the hierarchy as the length of the path increases. In the following confidence calculation, the result will be in the range $(-1, \frac{-5}{6}]$.

Definition 9 (Length of Path in a Hierarchy) *Given an axiom α of the form $\mathsf{CN} \sqsubseteq D$ in an ontology, where CN is a named concept. The length of the path of concept CN in the hierarchy is n. The confidence of α is defined as*

$$conf_{length}(\alpha) = \frac{1-n}{n},\ where\ n \geq 6$$

Since we expect longer paths to be more likely to have errors, the heuristic is only useful for lengthy hierarchies, and is not applicable for short ones. The threshold value, which is 6 by default, is somewhat arbitrary. From the experience of our subjects in the empirical studies, the length of a path less than 6 is easy to browse in ontology editors. Further work is necessary to find an optimal threshold value with respect to different types of ontologies.

6.3.8 Depth of Concepts in Hierarchy

When ontology users merge two satisfiable ontologies into a single one, the resulting ontology might be unsatisfiable. It is difficult to spot which axioms from which ontology are problematic. In this case, we assume that the more general axioms are more important. This heuristic has the following intuition: the concepts at the upper level are usually built for general purpose, so that more specific concepts can be constructed based on the upper ones.

This heuristic is particularly useful when integrating an ontology with a foundational (or upper) ontology. Foundational ontologies (e.g., Tambis.owl[3], DOLCE[4]) provide a structure upon which ontologies for specific subject matters can be based. It is important to ensure that integration preserves their semantics [33]. On the other hand, one of the typical scenarios in deployed Semantic Web applications is ontology reuse, where users build their own ontologies by extending existing ones, rather than starting from scratch [41]. The user may not necessarily understand the downloaded ontology (which is assumed to be satisfiable), and extends it with more specific concepts. However, inconsistencies in the ontology can easily occur due to contradictions between the definitions of newly added concepts and those of the original concepts. In the case of integrating with foundational ontologies or extending existing ones, some subjects assume that if we do not have information about which parts of the ontology are newly extended by the user, or which parts belong to the original, their strategies are to preserve the more general concept definitions, and hence the less specific ones are likely to be erroneous. This is captured by the following definition:

Definition 10 (Depth of a Concept) *Given an axiom α of the form $\mathsf{CN} \sqsubseteq D$, where CN is a named concept, $depth(\mathsf{CN})$ denotes the distance from CN to owl:Thing in the hierarchy. The*

[3]http://protege.stanford.edu/plugins/owl/owl-library/tambis-full.owl
[4]http://www.loa-cnr.it/DOLCE.html

confidence of α is defined as

$$conf_{depth}(\alpha) = \frac{1}{depth(\mathsf{CN})}$$

The results of the above confidence calculation will be in the range $(0, 1]$.

6.3.9 Linguistic Heuristic

Some subjects in the empirical studies believed that the name of a superconcept is often similar to its subconcepts. We measured the similarity of two strings on a scale from 0 to 1 based on Levenshtein's edit distance, $editDist$ [104]. The edit distance formulated by Levenshtein is a well-established method for weighting the difference between two strings. It measures the minimum number of token insertions, deletions, and substitutions required to transform one string into another using a dynamic programming algorithm. This technique is commonly used in ontology evaluation [22], ontology mapping [36], and ontology alignment [141]. We adapt the definition of lexical similarity measure for strings, Sim, which compares two lexical entries CN, DN, [106].

Definition 11 (Linguistic Heuristic) *Give an axiom α of the form $\mathsf{CN} \sqsubseteq \mathsf{DN}$, where both* CN *and* DN *are their local part of the URI in QName form or labels,*

$$Sim(\mathsf{DN}, \mathsf{CN}) = max(0, \frac{max(|\mathsf{CN}|, |\mathsf{DN}|) - editDist(\mathsf{CN}, \mathsf{DN})}{max(|\mathsf{CN}|, |\mathsf{DN}|)})$$

where $|\mathsf{CN}|$ is the length of the string

$$conf_{ling}(\alpha) = \begin{cases} Sim(\mathsf{DN}, \mathsf{CN}) & \textit{if Sim(CN,DN)} \geq \textit{0.4,} \\ 0 & \textit{if Sim(CN,DN)} < \textit{0.4} \end{cases}$$

Sim returns a degree of similarity between 0 and 1, where 1 stands for two identical strings and zero for completely different strings. We take the Sim value as the confidence value if it is equal to or larger than 0.4, otherwise, the confidence value is zero. This threshold value was obtained by trial and error. For example, *editDist*(PhDStudent, Student)=3. Sim(PhDStudent,Student) = $\frac{10-3}{10}$ = 0.7, the confidence of PhDStudent $\sqsubseteq$ Student is therefore 0.7.

6.3.10 Other Heuristics

We outlined heuristics for evaluating confidence above, and it is feasible that more may be added in future. In this section, we discuss other possible heuristics.

Size of Axioms. We can correlate the size of axioms with their confidence. The studies in Chapter 5 revealed that (1) if the ontology is built by a non-expert user, then longer axioms are usually non-intuitive, and unnecessarily complicated; they are likely to be erroneous compared to the shorter ones and hence should have lower confidence values; (2) if the ontology is built by experienced users, or extended from an existing one, then longer axioms are important and hence have higher confidence value.

Lexical-syntactic Patterns. Moreover, much research has been carried out in the area of ontology evaluation [22], where taxonomic relations and other semantic relations are evaluated by comparing against a source of data (e.g., a collection of documents) or by matching with lexical databases (e.g., WordNet [28]). The literature on ontology mapping also proposed linguistic techniques for ontology-mapping [150]. For example, subclass relations can be learnt from a semi-structured data source or Hearst patterns [68]. These methods can also be adapted to check the correctness of subclass relations, instantiation or disjointness etc. The work on ontology learning also combines linguistic analysis with statistical and/or machine learning approaches to learn and/or evaluate ontologies. For example, concept hierarchies are learnt from textual data, in which lexico-syntactic patterns are used as an effective means for extracting various types of lexical and ontological relationships such as hyponymy and meronymy [152; 151; 30].

To debug ontologies, similar approaches can be adopted to evaluate the correctness of axioms through WordNet or Google. We can obtain the hypernymy, synonym and antonym information from WordNet. As the World Wide Web is the currently biggest existing source of common sense knowledge [152], we can acquire knowledge from the Web using Google^{TM} API, where we search for lexico-syntactic patterns [68] indicating a certain semantic or ontological relation and count their occurrences in the Web. The statistics distribution of these patterns is used to calculate the confidence of the correctness of the subsumption relations.

To evaluate confidence of an axiom Student $\sqsubseteq$ Person, we can construct Google queries based on the following lexico-syntactic patterns [68; 29]:

1. NP_0 such as $\{NP_1 \cdots$ (and/or)$\}$ NP_n [5]
   e.g., person such as staff or student;
2. such NP_0 as $\{NP_1 \cdots$ (or/and)$\}$ NP_2
   e.g., such a person as a student or a lecturer;
3. $\{NP_1 \cdots NP_n$ (and/or)$\}$ other NP_0
   e.g., lecturers or students or other persons;
4. NP_0 $\{,\}$ (including/especially) $\{NP_1 \cdots$ (and/or)$\}$ NP_n;
5. NP_1 (be) (a/an) NP_0;
6. NP_1, another NP_0;
7. NP_0 like NP_1

We then submit the queries to the Google API, and the hit counts for all lexico-syntactic patterns are obtained; this gives evidence for the existence of a relation, we add the hits on these queries (i.e., the number of results returned by the search engine) to give evidence that Student is a subclass of Person. The following definition formalises the above intuition.

Definition 12 (Pattern Matching) *Give an axiom α of the form* CN$\sqsubseteq$ DN, *where both* CN *and* DN *are their local part of the URI in QName form or labels,*

$$conf_{query}(\alpha) = \frac{\sum_{q \in Q} |Hits(q)|}{|Hits(\mathsf{CN} \wedge \mathsf{DN})|}$$

[5]NP_i stands for a noun phrase

where Q is a set of queries, $Hits(q)$ is the hits on a query q, $Hits(\mathsf{CN} \wedge \mathsf{DN})$ is the number of hits returned by the search engine for the concepts CN *and* DN.

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

Definition 13 *Let α be an axiom, the reliability rating of the information source of the axiom is defined as*

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

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

- *Case 1: the information is reliable,* $r = +1$
  In this case it is assumed that newer information (newly added or modified axioms) generally reflects a more accurate view of the domain (the Principle of Primacy of New Information [34]). This means the axioms which are newly added or modified are usually more accurate than those provided earlier and have higher confidence.

- *Case 2: the information is unreliable,* $r = -1$
  In this case the effect of recency is the inverse of the above. The new information can be rejected, because the new information is obtained from unreliable and untrustworthy sources. This means the newly added axioms have lower confidence, and older axioms, having stood the test of time, have higher confidence.

- *Case 3: the information is unknown,* $r = 0$
  If the reliability of the information source is unknown, then the confidence of axioms is dependent on the frequency of modifications, rather than its reliability. Axioms which are frequently updated are likely to be vulnerable to further modifications, as the users may have been making experimental changes. Therefore, the confidence of an axiom is inversely proportional to the frequency of its modifications.

Definition 14 *Let t be the length of time since the last modification on an axiom α, r be the reliability value of the information provided by users, and $modify(\alpha)$ be the number of modifications on the axiom α, e^x be the exponential function. The confidence of the axiom is defined as*

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

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

The above three heuristics cannot be currently evaluated by our usability study in Section 8.2, because we lack a sufficient number of existing unsatisfiable ontologies. We report on them because we believe that they constitute a potentially interesting direction of future work. For evaluating the 'size of axioms' heuristic, we need a large number of real world ontologies, which contain various sizes of axioms. For evaluating the usage heuristic, it is difficult to request a number of subjects to use the ontologies over a period of time, so as to keep trace of their modification patterns in our study. To obtain the reliability of the information source, more work is needed to build and maintain *a web of trust* [58]. Still, these heuristics are applicable in different scenarios. Firstly, an ontology which describes a certain domain of interest inevitably evolves in the course of its lifetime. Changes in an ontology may be made when the domain of interest changes, the user requirements change, or new information is available for extension [42]. In such cases, the newly added axioms usually have higher reliability, and hence we have higher confidence in them (see case 1 in Definition 14 in Section 6.3.10). Secondly, in collaborative ontology building scenarios, it is reasonable to assign higher confidence to the local axioms over axioms from imported ontologies. With the owl:imports construct, we cannot import a certain interesting part of the ontology and leave out the irrelevant parts. Therefore, the imported axioms which are contradictory to the existing ones have low reliability and, hence, we have lower confidence in them (see case 2 in Definition 14 in Section 6.3.10).

6.4 Fine-grained Approach to Ranking Parts of Axioms

This chapter describes a heuristic approach to selecting appropriate axioms for removal or modification, in order to resolve a concept's unsatisfiability. In fact, as mentioned in Section 4.1, it often happens that only certain parts of axioms are responsible for the unsatisfiability. Therefore, it is more appropriate to evaluate the confidence of axiom components, instead of whole axioms, so that the user is advised on which parts of axioms are more likely to be less important or correct. In the following, we discuss how the heuristics capturing concept component level instead of at the coarse grained axiom level:

- The 'impact of removal' heuristic already captures the impact of change on the ontology due to the removal of (parts of) axioms

- The ‘disjointness’ and ‘linguistic’ heuristics only consider disjointness/subsumption relationships between named concepts. They are applicable at the concept component level.

- The ‘sibling patterns’ heuristic can be further refined so that it can evaluate the confidence values of concept and role names in axioms. For example, given an axiom, $A \sqsubseteq \forall R.C \sqcap \exists R^*.(C^* \sqcap D^*) \sqcap B$, where concept/role names tagged with (*) indicate that they are responsible for a concept’s unsatisfiability, the heuristic in Section 6.3.6 compares the problematic part $\exists R^*.(C^* \sqcap D^*)$ with A’s sibling concepts. Obviously, this heuristic can be more precise. It would be helpful to assign individual concept and role names with confidence values, so that the user can modify the precise problematic parts of axioms.

- The ‘length of path’, ‘depth of concepts’ heuristics calculate the confidence of axioms depending on concept hierarchies. Axioms of the form $\mathsf{CN} \sqsubseteq D$ or $\mathsf{CN} \doteq D$ are considered; their confidence values only depend on the location of CN in the concept hierarchy, but not on the problematic parts of D. Therefore, the heuristics are more useful in concept hierarchies, but limited to complex axioms.

6.5 Related Work

In this section, we compare our heuristic approach with existing work which provides strategies for ranking axioms, and then discuss the use of common error patterns.

6.5.1 Ranking Axioms

The most relevant work is SWOOP [91], in which the authors propose various strategies to rank erroneous axiom(s) to be removed from the MUPSs in order to fix the unsatisfiable concepts. We now compare their strategies with our approach:

1. *Arity* of axioms – Given arity n, SWOOP assigns confidence $-kn$ to the axiom, for some constant k. In this chapter, we have used a different formula for this factor, $(1-n)/n$, which is normalised into the range $(-1, 0]$. Currently, we cannot conclude which formula is more effective; a further study is needed.

2. *Impact* of removing axioms on the ontology – SWOOP ranks axioms based on the number of entailments (subsumption relations between named concepts, instantiations of named concepts) which would be lost if the whole axioms are removed. This implies that an axiom which breaks more relationships is more important in the ontology. In comparison, we implement a finer-grained approach which allows us to calculate the loss of entailments of atomic concepts due to removal of parts of axioms. The lost entailments are calculated in three ways: the named concepts involved in the unsatisfiability, the satisfiable named concepts irrelevant to the unsatisfiability, as well as subsumption relations between named concepts (Chapter 7 provides the details). We also consider the number of named concepts which are affected by the removal of (parts of) axioms, in addition to the number of lost entailments.

3. Relevance to the ontology in terms of its *usage* – in SWOOP the relevance of an element to the ontology depends on the usage of the element in the ontology. This means, if a concept is highly referenced by other elements in an ontology, then it is more relevant to the ontology, and hence more important. The usage of an axiom is the summation of the number of all ontology elements which refer to the elements of the axiom. For example, given an axiom $\mathsf{Bird} \sqsubseteq \mathsf{Animal}$, if Animal has twenty direct subconcepts, then the usage of the axiom is twenty, thus the axiom is quite important due to its high usage. However, when this axiom is removed, the other subconcepts of Animal are completely unaffected. We believe that the usage of an axiom should only take its left-hand side into account, i.e., only the number of axioms which refer to Bird are counted. If Bird has ten direct subconcepts, then its usage will be ten. However, by only considering the left-hand side of axioms, this calculation is similar to SWOOP's impact of removal and hence the 'usage' factor is redundant.

6.5.2 Common Error Patterns

Rector et al. [125] list a number of common errors in modelling OWL ontologies (shown in Table 6.1); Kalyanpur et al. [91] further extend the list, and keep a library of common errors in modelling OWL ontologies. Kalyanpur et al. check if any of the axioms has a pattern corresponding to one in the library, and if so, suggest the intended axiom to the user as a replacement. For example, using this approach, an equivalent axiom ($A \doteq C$) is suggested to be changed to be a subsumption axiom ($A \sqsubseteq C$). However, their description of rewriting suggestions is sketchy [91]. If all axioms in the MUPS of an unsatisfiable concept match the error patterns, it is not clear which axiom should be rewritten. Also, if a problematic axiom matches multiple error patterns, it is not clear what the rewriting suggestions are. For example, given an axiom $A \doteq \forall R.(C \sqcap D) \sqcap \forall S.(\neg C \sqcap D)$, this axiom matches three error patterns in the library. It is challenging to make concrete suggestions for how to resolve this problem, because there are often a lot of options for rewriting axioms, but usually only a few of these options are intuitively acceptable [105].

In comparison, as presented in Section 6.3.6, we make use of these common errors for evaluating the correctness of axioms, in addition to the pattern of siblings. In brief, if an axiom (the left-hand side of the axiom is a named concept CN) matches the common error pattern, but CN has no sibling for comparison, then we keep the axiom's default confidence value. On the other hand, if CN has many sibling concepts which match the pattern of 'intended' axioms, then we say this axiom is more likely to be erroneous.

6.6 Conclusion

In this chapter we aim to indicate which problematic axiom(s) should be selected for removal/moidification, in order to help users resolve unsatisfiable concepts. The problematic axioms are ranked in order of confidence. The confidence value of axioms is calculated from a set of heuristics. These heuristics are formalised and grouped into (1) impact of removal on the ontology, (2) knowledge of ontology structure, and (3) linguistic heuristic. We have conducted a usability evaluation to demonstrate the usefulness of the heuristics in practice. The results are presented in Chapter 8.

Chapter 7

Impact of Changes on the Ontology

The previous chapter describes a heuristics-based service to select appropriate axioms for removal or repair, in order to resolve unsatisfiable concepts. However, it is still not clear what is the impact of changes on the ontology. In this chapter, we go a step further by utilising the fine-grained approach introduced in Chapter 4. Our aim is to minimise the impact of changes and prevent unintended entailment loss, in the case of removal. To do so, we proposed *helpful* and *harmful* changes, those which are helpful restore lost entailments (due to axiom removal), and those that are harmful cause additional unsatisfiability.

Furthermore, in order to increase the efficiency of satisfiability checking when updating axioms, we make use of the cached results from previous runs of the tableau algorithm to check the (un)satisfiability of concepts, without running the tableau algorithm from scratch.

The rest of this chapter is organised as follows. Section 7.1 gives the overview of the chapter. The impact of removing axioms is described in Section 7.2. The methods of identifying harmful and helpful changes are presented in Section 7.3. The proposed caching technique is described in Section 7.4. The chapter closes with a comparison with related work and some conclusions.

7.1 Overview

In Chapter 6, we use a set of heuristics to evaluate the confidence of axioms so as to suggest to the user which axioms are appropriate to be removed or modified. After evaluating the problematic axioms, we are able to select which problematic axiom(s) to modify based on the confidence we have in the axioms. The next step is to rewrite the erroneous axiom(s), rather than to directly remove them from the ontology. It frequently happens that rewriting a problematic axiom might not resolve the target unsatisfiable concept, but might introduce unintended consequences, such as additional unsatisfiabilities, unintended new/lost entailments, and such consequences can be far from obvious [52].

In this chapter, we utilise the fine-grained approach introduced in Chapter 4. This approach allows us to make the following two contributions. The first is to calculate the lost entailments of named concepts due to the removal of axioms. Whenever (parts of) an axiom is removed, it frequently happens that indirect or implicit entailments are lost. In order to minimise the impact on the ontology, we analyse the lost entailments (e.g., concept subsumption) of named concepts which occur due to the removal of (parts of) axioms. The second contribution is to identify harmful and helpful changes; this is where the fine-grained tracing information is useful to facilitate rewriting the problematic axioms, rather than removing them completely. It should be noted that improperly rewriting a problematic axiom might not resolve the unsatisfiability, and could introduce additional unsatisfiable concepts into the ontology. For this purpose we define *harmful* and *helpful* changes with respect to a concept. A harmful change cannot resolve the problem, or might cause additional unsatisfiable concepts in the ontology; a helpful change resolves the problem without causing additional contradictions, and restores some lost entailments.

Typically, whenever an axiom is changed, a DL reasoner has to perform a consistency check on the ontology from scratch so as to calculate the impact of the change on the ontology. For ontologies with huge numbers of concepts/axioms, it takes a long time to check whenever a change is made. To address this limitation, we utilise a caching technique to optimise the consistency checking of an ontology after updating axioms. We reuse the results of an earlier tableau algorithm run to check the satisfiability of concepts without running the tableau algorithm on the whole ontology from scratch. A considerable amount of work can be saved when larger or complex ontologies are modified in this way.

7.2 Impact of removing axioms

After the parts of the axioms causing the unsatisfiability of concept(s) are identified, the next step is to resolve the unsatisfiability. In this section, we discuss, with examples, the impact of removing axioms on an ontology.

The simplest way to resolve unsatisfiability is to remove parts of the problematic axioms or the whole axioms. However, in this case, it will be easy for ontology modellers to accidentally remove indirect or implicit entailments in the ontology. We use the following mad_cow[1] example to explain what we mean by the impact of removing axioms from an ontology:

Example 33 *Given an ontology where* Mad_Cow *is unsatisfiable due to axioms* $\alpha_1, \alpha_3, \alpha_4, \alpha_5$, *the concepts and roles tagged with a star (*) are responsible for the unsatisfiability:*

α_1: $\mathsf{Mad\_Cow}^* \doteq \exists\, \mathsf{eats}^*.((\exists \mathsf{part\_of}^*.\mathsf{Sheep}^*) \sqcap \mathsf{Brain}) \sqcap \mathsf{Cow}^*$

α_2: $\exists \mathsf{part\_of.Plant} \sqcup \mathsf{Plant} \sqsubseteq \neg(\exists \mathsf{part\_of.Animal} \sqcup \mathsf{Animal})$

α_3: $\mathsf{Cow}^* \sqsubseteq \mathsf{Vegetarian}^*$

α_4: $\mathsf{Vegetarian}^* \doteq \forall\, \mathsf{eats}.(\neg\, \mathsf{Animal}) \sqcap \mathsf{Animal} \sqcap \forall \mathsf{eats}^*.(\neg\exists \mathsf{part\_of}^*.\mathsf{Animal}^*)$

α_5: $\mathsf{Sheep}^* \sqsubseteq \forall\, \mathsf{eats.Grass} \sqcap \mathsf{Animal}^*$

α_6: $\mathsf{Grass} \sqsubseteq \mathsf{Plant}$

α_7: $\mathsf{Giraffe} \sqsubseteq \mathsf{Vegetarian}$

When an axiom involved in the unsatisfiability of a concept is changed, we calculate the impact of removal on the ontology in three ways:

1. *The named concepts involved in the unsatisfiability*: These concepts might lose entailments which are not responsible for the unsatisfiability. To resolve Mad_Cow, one may claim that not all cows are vegetarians if there exist mad cows, therefore, α_3 is removed. However, the indirect assertions $\mathsf{Mad\_Cow} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg\, \mathsf{Animal})$ and $\mathsf{Cow} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg\, \mathsf{Animal})$ will be lost, as we know the part $\mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg\, \mathsf{Animal})$ in α_4

[1]http://www.cs.man.ac.uk/∼horrocks/OWL/Ontologies/mad_cows.owl

is not relevant to Mad_Cow's unsatisfiability. Similarly, if α_4 is removed, then the indirect assertion Vegetarian $\sqsubseteq$ Animal $\sqcap$ $\forall$eats.($\neg$ Animal) and the above two assertions are all lost.

2. *The satisfiable named concepts irrelevant to the unsatisfiability*: Other named concepts irrelevant to the satisfiability might lose entailments introduced by the axiom to be changed. The entailments we consider in this case are indirectly asserted in the ontology before the change. If the user removes the problematic part $\forall$eats.($\neg\exists$ part_of.Animal) from α_4, then all the subconcepts of Vegetarian will be affected. The indirect assertion all giraffes only eat something which is not part of an animal inherited from Vegetarian will be lost. Cow, which is involved in the unsatisfiability, is not considered here, as the assertion all cows only eat something which is not part of an animal will still make Mad_Cow unsatisfiable;

3. *The classification of the named concepts of the ontology*: The satisfiable named concepts might lose implicit subsumption relations due to the change of axioms. We run classification on the example, and find that Sheep is subsumed implicitly by Vegetarian due to axioms $\alpha_2, \alpha_4, \alpha_5, \alpha_6$. The change of α_4 might also remove the inferred subsumption relationship between Sheep and Vegetarian.

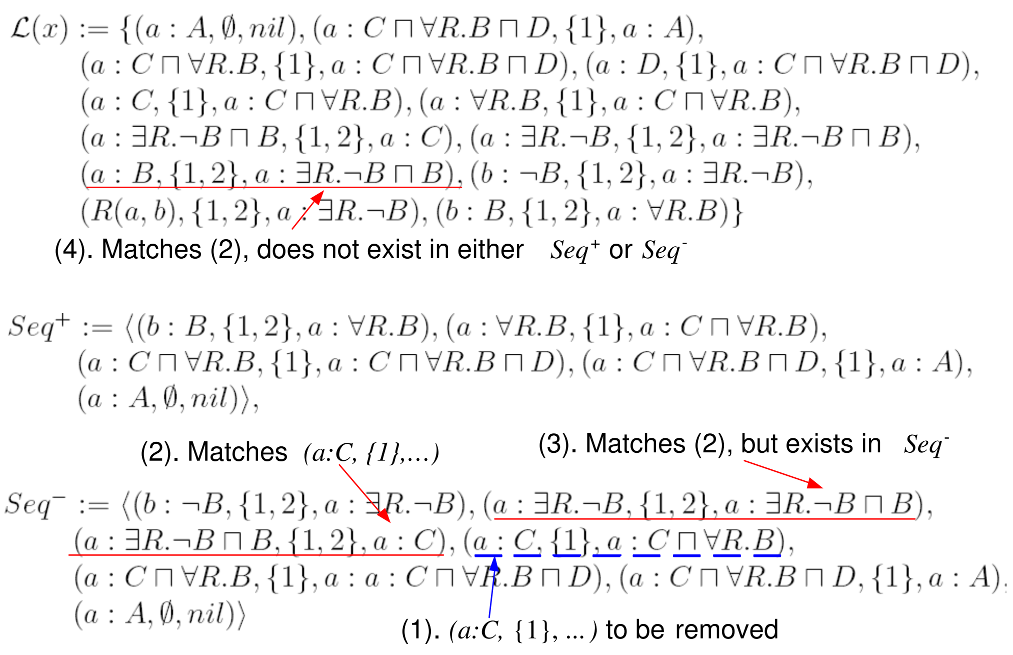
We now deal with each of the above three cases.

7.2.1 Impact on named concepts involved in the unsatisfiability

We first describe how to calculate the impact of the removal of (parts of) axioms on the named concepts involved in the unsatisfiability. In the following we use the Example 25 described in Chapter 4 (reproduced below as Example 34) to explain how to find lost entailments, which are not responsible for the concept's unsatisfiability, by analysing the sequences of the clashes of an unsatisfiable concept.

Example 34 *Let us assume that an ontology $\mathcal{O}$ contains the following axioms:*
α_1: $A \doteq C \sqcap \forall R.B \sqcap D$
α_2: $C \doteq \exists R.\neg B \sqcap B$
α_3: $G \doteq \forall R.(C \sqcap F)$
α_4: $K \doteq C \sqcap \forall R.(P \sqcap F)$
α_5: $P \doteq \forall R.F \sqcap B$

Our idea is to search for any element which exists in the fully expanded tree but not in the sequences of the clashes. In Example 34, if C in axiom α_1 is going to be removed (i.e., C is replaced by $\top$ in α_1), then we have to calculate the lost entailments of A which are not responsible for A's unsatisfiability. Figure 7.1 shows the fully expanded tree for A and the sequences of the clash in A. We find that $(a : C, \{1\}, a : \forall R.B)$ exists in Seq^- (cf. 1 in Figure 7.1). We search for elements in the tree whose second concept assertion is $a : C$, but which do not exist in either Seq^+ or Seq^-. $(a : \exists R.\neg B \sqcap B, \{1, 2\}, a : C)$ matches $a : C$ but exists in Seq^- (cf. 2), so we keep searching for other elements whose second concept assertion is $a : \exists R.\neg B \sqcap B$. The matched



$$\begin{aligned}\mathcal{L}(x) := \{&(a : A, \emptyset, nil), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),\\ &(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : D, \{1\}, a : C \sqcap \forall R.B \sqcap D),\\ &(a : C, \{1\}, a : C \sqcap \forall R.B), (a : \forall R.B, \{1\}, a : C \sqcap \forall R.B),\\ &(a : \exists R.\neg B \sqcap B, \{1,2\}, a : C), (a : \exists R.\neg B, \{1,2\}, a : \exists R.\neg B \sqcap B),\\ &\underline{(a : B, \{1,2\}, a : \exists R.\neg B \sqcap B)}, (b : \neg B, \{1,2\}, a : \exists R.\neg B),\\ &(R(a,b), \{1,2\}, a : \exists R.\neg B), (b : B, \{1,2\}, a : \forall R.B)\}\end{aligned}$$

$$\begin{aligned}Seq^+ := \langle &(b : B, \{1,2\}, a : \forall R.B), (a : \forall R.B, \{1\}, a : C \sqcap \forall R.B),\\ &(a : C \sqcap \forall R.B, \{1\}, a : C \sqcap \forall R.B \sqcap D), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),\\ &(a : A, \emptyset, nil)\rangle,\end{aligned}$$

$$\begin{aligned}Seq^- := \langle &(b : \neg B, \{1,2\}, a : \exists R.\neg B), \underline{(a : \exists R.\neg B, \{1,2\}, a : \exists R.\neg B \sqcap B)},\\ &\underline{(a : \exists R.\neg B \sqcap B, \{1,2\}, a : C)}, (\underline{a} : \underline{C}, \underline{\{1\}}, \underline{a} : \underline{C} \underline{\sqcap} \underline{\forall R}.\underline{B}),\\ &(a : C \sqcap \forall R.B, \{1\}, a : a : C \sqcap \forall R.B \sqcap D), (a : C \sqcap \forall R.B \sqcap D, \{1\}, a : A),\\ &(a : A, \emptyset, nil)\rangle\end{aligned}$$

(4). Matches (2), does not exist in either Seq+ or Seq-
(2). Matches (a:C, {1},...)
(3). Matches (2), but exists in Seq-
(1). (a:C, {1}, ...) to be removed

Figure 7.1: The fully expanded tree for A in Example 34

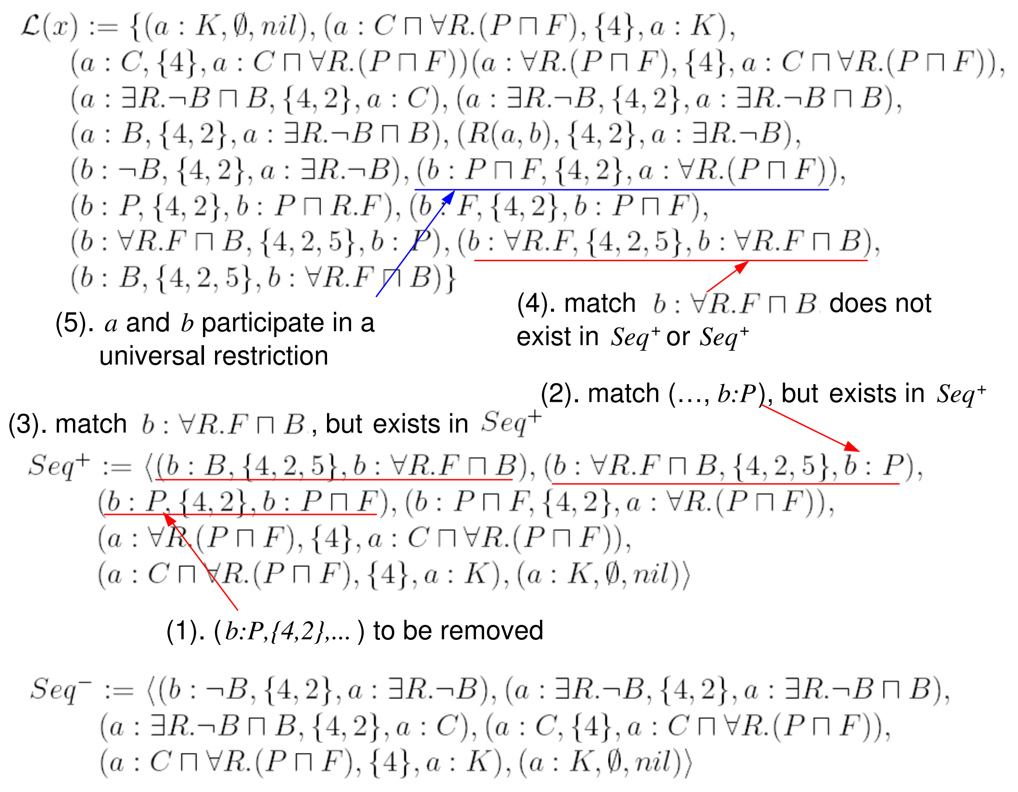
elements are $(a : \exists R.\neg B, \{1,2\}, a : \exists R.\neg B \sqcap B)$ which exists in Seq^- (cf. 3) and $(a : B, \{1,2\}, a : \exists R.\neg B \sqcap B)$ (cf. 4) which does not exist in Seq^+ or Seq^- and has the same individual as $(a : A, \emptyset, nil)$. This means that B is a superconcept of A, and hence, the lost entailment is $A \sqsubseteq B$.

Continuing Example 34, to resolve the unsatisfiability of K, if concept P in axiom α_4 is removed, then the lost entailment is $K \sqsubseteq \forall R.(\forall R.F)$. We describe how to find lost entailments with role relationships as follows. Figure 7.2 shows the fully expanded root node for K, and two sequences of the clash in the node.

We find that $(b : P, \{4,2\}, b : P \sqcap F)$ exists in Seq^+ (cf. 1 in Figure 7.2). We search for elements in the tree whose second concept assertion is $b : P$, but which do not exist in Seq^+ or Seq^-. $(b : \forall R.F \sqcap B, \{4,2,5\}, b : P)$ matches $b : P$ but exists in Seq^+ (cf. 2), so we keep searching for other elements whose second concept assertion is $b : \forall R.F \sqcap B$. The matched elements are $(b : B, \{4,2,5\}, b : \forall R.F \sqcap B)$ which exists in Seq^+ (cf. 3) and $(b : \forall R.F, \{1,2\}, b : \forall R.F \sqcap B)$ (cf. 4) which does not exist in Seq^+ or Seq^-. That means $(b : \forall R.F, \{1,2\}, b : \forall R.F \sqcap B)$ is not responsible for the unsatisfiability of K. However, the individual of the element is b which is different from that of $(a : K, \emptyset, nil)$ in the label of the node. We have to search for the relationship between the two individuals, a and b.

We firstly search for the elements in Seq^+ succeeding $(b : P, \{4,2\}, b : P \sqcap F)$ whose first and second individuals are different. In the example, $(b : P \sqcap F, \{4,2\}, a : \forall R.(P \sqcap F))$ is matched, it shows that a is related to b via a universal restriction $\forall R.-$ (cf. 5). Therefore, we can know that the lost entailment is $K \sqsubseteq \forall R.(\forall R.F)$.

Assume that A is an unsatisfiable named concept, and α_i is involved in its unsatisfiability, and there exists a clash in node x in the fully expanded tree. When a concept C on the right-hand side of α_i is to be removed, we can calculate the lost entailments of A with the algorithm shown in Figure 7.3.



$$
\begin{aligned}
\mathcal{L}(x) := \{&(a:K,\emptyset,nil),(a:C\sqcap\forall R.(P\sqcap F),\{4\},a:K),\\
&(a:C,\{4\},a:C\sqcap\forall R.(P\sqcap F))(a:\forall R.(P\sqcap F),\{4\},a:C\sqcap\forall R.(P\sqcap F)),\\
&(a:\exists R.\neg B\sqcap B,\{4,2\},a:C),(a:\exists R.\neg B,\{4,2\},a:\exists R.\neg B\sqcap B),\\
&(a:B,\{4,2\},a:\exists R.\neg B\sqcap B),(R(a,b),\{4,2\},a:\exists R.\neg B),\\
&(b:\neg B,\{4,2\},a:\exists R.\neg B),\underline{(b:P\sqcap F,\{4,2\},a:\forall R.(P\sqcap F))},\\
&(b:P,\{4,2\},b:P\sqcap R.F),(b:F,\{4,2\},b:P\sqcap F),\\
&(b:\forall R.F\sqcap B,\{4,2,5\},b:P),\underline{(b:\forall R.F,\{4,2,5\},b:\forall R.F\sqcap B)},\\
&(b:B,\{4,2,5\},b:\forall R.F\sqcap B)\}
\end{aligned}
$$

(5). a and b participate in a universal restriction
(4). match $b:\forall R.F\sqcap B$ does not exist in Seq^+ or Seq^+
(2). match (..., b:P), but exists in Seq^+
(3). match $b:\forall R.F\sqcap B$, but exists in Seq^+
(1). (b:P,{4,2},...) to be removed

$$
\begin{aligned}
Seq^+ := \langle&\underline{(b:B,\{4,2,5\},b:\forall R.F\sqcap B)},\underline{(b:\forall R.F\sqcap B,\{4,2,5\},b:P)},\\
&\underline{(b:P,\{4,2\},b:P\sqcap F)},(b:P\sqcap F,\{4,2\},a:\forall R.(P\sqcap F)),\\
&(a:\forall R.(P\sqcap F),\{4\},a:C\sqcap\forall R.(P\sqcap F)),\\
&(a:C\sqcap\forall R.(P\sqcap F),\{4\},a:K),(a:K,\emptyset,nil)\rangle
\end{aligned}
$$

$$
\begin{aligned}
Seq^- := \langle&(b:\neg B,\{4,2\},a:\exists R.\neg B),(a:\exists R.\neg B,\{4,2\},a:\exists R.\neg B\sqcap B),\\
&(a:\exists R.\neg B\sqcap B,\{4,2\},a:C),(a:C,\{4\},a:C\sqcap\forall R.(P\sqcap F)),\\
&(a:C\sqcap\forall R.(P\sqcap F),\{4\},a:K),(a:K,\emptyset,nil)\rangle
\end{aligned}
$$

Figure 7.2: The fully expanded tree for the satisfiability test of K

7.2.2 Impact on satisfiable named concepts irrelevant to the unsatisfiability

We now describe how to calculate the impact on named concepts irrelevant to the unsatisfiability. In this case, we only consider the lost entailments of satisfiable named concepts. Note that when a concept is unsatisfiable, it is trivially a subconcept of all satisfiable concepts and equivalent to all unsatisfiable concepts. If an axiom $C \sqsubseteq D$ is removed, any named concept in other axioms, which refers to C, will lose entailments introduced by this axiom. In general we lose $X \sqsubseteq Y$ where X is a subconcept of C and Y is a superconcept of D. Continuing the mad_cow example, when the problematic part $\forall$ eats.($\neg\exists$ part_of.Animal) from α_4 is removed, all the subconcepts of Vegetarian which are not relevant to the unsatisfiability will be affected. It is obvious that the lost entailment of Giraffe is Giraffe $\sqsubseteq \forall$ eats.($\neg\exists$ part_of.Animal).

For those named concepts which refer to a concept to be removed not just via subsumption relations, the lost entailments cannot be as easily obtained as above. For the mad_cow example, if α_4 is changed to be Vegetarian$\doteq$ $\forall$ eats.Plant $\sqcap$ Animal $\sqcap$ $\forall$eats.($\neg\exists$part_of.$\bot$), then we cannot say the lost entailment is Vegetarian $\sqsubseteq$ $\forall$eats.($\neg\exists$part_of.Animal). This is because the definition of Vegetarian still implies that it only eats part of anything, which includes $\neg$Animal.

The lost entailment of such concepts can be computed by calculating the difference between the original and modified concepts. To do this we adapt the notion of the "*difference*" operator between concepts which is defined in [145]. The difference between C and C' (1) contains enough information to yield the information in C if added to C', i.e., it contains all information from C which is missing in C', and (2) is maximally general, i.e., it does not contain any additional unnecessary information.

```
    Given: an unsatisfiable $A$, the sequences of $Seq^+$, $Seq^-$ of a clash,
           the label of node $x$ is $\mathcal{L}(x)$, and $C$ is to be removed from $\alpha_i$
1.  let $a$ be the individual of the last element of the $Seq^+$;
2.  $lostEnt := \{\}$;
3.  $setEle := \{\}$;
4.  $roleSeq := \langle\rangle$;
5.  $ele := SearchSequence((- : C, -, -), Seq^+)$, where $ele = (a' : C, -, -)$
                //search for the element in the sequence whose first concept is $C$
6.  if $(ele$ != null)     then $Seq := Seq^+$;
7.  else $ele := SearchSequences((- : C, -, -), Seq^-)$, where $ele = (a' : C, -, -)$
8.       $Seq := Seq^-$;
9.  end if
10. $\mathcal{S} := SearchElement((-, -, a' : C), \mathcal{L}(x), setEle)$
11. for each $\varepsilon \in \mathcal{S}$, where $\varepsilon = (a_1 : D_1, -, -)$
12.     if $(a = a_1)$, then
13.         $lostEnt := lostEnt \cup \{A \sqsubseteq D_1\}$;
14.     else
15.         $roleSeq := SearchRoleSeq((a' : C, -, -), Seq, roleSeq, a_1)$;
16.         $lostEnt := lostEnt \cup \{createSubsumption(A, roleSeq, D_1)\}$;
                //$createSubsumption$ creates a subsumption relationship for $A$,
                //e.g., if $roleSeq = \langle\forall R, \exists R\rangle$, then an entailment $A \sqsubseteq \forall R.(\exists R.D_1)$ is created.
17. end for
18. return $lostEnt$;

19. subroutine: $SearchElement((-, -, a' : C), \mathcal{L}(x), setEle)$
20. $\mathcal{S} := search((-, -, a' : C), \mathcal{L}(x))$;
                //search for elements in $\mathcal{L}(x)$ whose second concept is $C$
21. for each $\varepsilon \in \mathcal{S}$, where $\varepsilon = (b : D_1, -, a' : C)$
22.     if $\varepsilon$ exists in $Seq^+$ or $Seq^-$, then
23.         $setEle := setEle \cup SearchElement((-, -, b : D_1), \mathcal{L}(x), setEle)$
24.     else    $setEle := setEle \cup \{\varepsilon\}$;
25. end for
26. return $setEle$;

27. subroutine: $SearchRoleSeq((a' : C, -, -), Seq, roleSeq, a_1)$
28. $\varepsilon := searchSuccessor((a' : C, -, -), Seq)$, where $\varepsilon = (a' : -, -, b : E)$, $a' \neq b$
        //search for the first element succeeding ($a'$ : $C$,-,-) in the $Seq$ with different individuals
29. if ($\varepsilon$ = null), then
30.     $\varepsilon := searchPredecessor((a' : C, -, -), Seq)$, where $\varepsilon = (b : E, -, a' : -)$, $a' \neq b$
        //search for the first element preceding ($a'$ : $C$,-,-) in the $Seq$ with different individuals
31. end if
32. if $E$ of the form $\forall R.-$, then
33.     $roleSeq := roleSeq \cdot \langle\forall R\rangle$;    //where $\cdot$ means to append an element to a sequence
34. else $roleSeq := roleSeq \cdot \langle\exists R\rangle$;
35. if $(a_1 = b)$, then
36.     return $roleSeq$;
37. else     return $SearchRoleSeq((b : E, -, -), Seq, roleSeq, a_1)$;
```

Figure 7.3: Algorithm for Finding Lost Entailments

Definition 15 (Difference of Concepts) *Let C and C' be the original and modified concept expressions, the difference between C and C', which is a set of concepts, is defined as*

$$\mathit{difference}(C, C') = \begin{cases} max_{\sqsupseteq}\{E|E \doteq C \sqcup \neg C'\} & \text{if } C \sqsubseteq C', \\ max_{\sqsupseteq}\{E|E \doteq C' \sqcup \neg C\} & \text{if } C' \sqsubseteq C \end{cases}$$

We reproduce Example 34 as follows:
α_1: $A \doteq\ C \sqcap \forall R.B \sqcap D$
α_2: $C \doteq\ \exists R.\neg B \sqcap B$
α_3: $G \doteq\ \forall R.(C \sqcap F)$

If $\exists R.\neg B$ in axiom α_2 is removed, then the modified axiom becomes $C \sqsubseteq B$. As α_3 refers to C, the lost entailment of G will be $\forall R.(\exists R.\neg B \sqcap B \sqcap F) \sqcup \neg\forall R.(B \sqcap F)$, i.e., $\forall R.(\exists R.\neg B) \sqcup \neg\forall R.(B \sqcap F)$. The disadvantage of this calculation is that the representation of lost entailments could be too complicated for human users to understand, the simplification of such representations is therefore necessary in the future. Brandt et al. [21] introduced a syntax-oriented difference operator, but the algorithm only supports the difference between an $\mathcal{ALC}$- and an $\mathcal{ALE}$-concept description. As $\mathcal{ALE}$ does not support disjunction concepts, their difference operator is not applicable to our approach.

7.2.3 Impact on the Classification

Besides deciding the satisfiability of concept descriptions, description logic reasoners are able to compute the classification of an ontology. Classification is the process of determining the subsumption relationship between any two named concepts in an ontology; e.g., for two named concepts A and B, it determines whether $A \sqsubseteq B$ and/or $B \sqsubseteq A$. Recall that reasoners decide subsumption relationships by reducing the problem to a satisfiability test (i.e., $A \sqcap \neg B$ is unsatisfiable if $A \sqsubseteq B$ holds). Whenever an axiom is changed, the classification of the ontology might be affected. We aim to point out to the user which parts of the classification will be affected if a certain change is made to the ontology. If the classification of the entire ontology must be checked after each change, then it will involve n^2 subsumption tests for n named concepts; moreover, each subsumption test (checking for satisfiability in $\mathcal{ALC}$ w.r.t. general inclusion axioms) is EXPTIME-complete [134]. It is impractical to run this classification test after each change made to the ontology. In this section, we will describe how we make use of the sequences of clashes (satisfiability test) to check if existing subsumption relations will be affected. If it is not affected, the subsumption test can be skipped.

Due to the monotonicity of the DLs we consider in this thesis, removal of (part of) axioms cannot add new entailments, and will not change any previous non-subsumption relationships. Therefore, we only need to re-check if the removal of axioms will invalidate the previously found subsumption relationships. By building a tree with the application of the expansion rules on $A \sqcap \neg B$, we can obtain the sequences of the clashes. The elements in the sequences are the cause of the unsatisfiability, that is the subsumption relationship. With the sequences of clashes in the tree, we can analyse if a certain removal/change of (part of) an axiom will affect the current subsumption relation. Therefore, we are able to predict which subsumption relationships of named concepts will be affected, and skip the subsumption tests for the unaffected named concepts.

We check if a concept component of an axiom which is going to be removed will affect the previously found subsumption as follows:

Lemma 3 *Given a terminology $\mathcal{T}$ such that $\mathcal{T} \models A \sqsubseteq B$, where A and B are named concepts, and a fully expanded tree* $\mathbf{T}$ *of $A \sqcap \neg B$, the sequences of clashes in the tree are obtained. Let I_u be the union of the index-set of the first element in all of the sequences. Assume that a concept component C on the right-hand side of α_i is going to be removed, where $\alpha_i \in \mathcal{T}$, the subsumption $A \sqsubseteq B$ is unaffected if either one of the following conditions hold:*

1. *$i \notin I_u$, α_i is not involved in the unsatisfiability,*

2. *$(- : C, I, -)$ and $(- : C', I, -)$ do not exist in any sequences of clashes where $i \in I$, $i \in I_u$ and C' is the negated C*

Proof The sequences of the clashes in $\mathbf{T}$ contain the concept components and sets of axioms which are relevant to the subsumption.

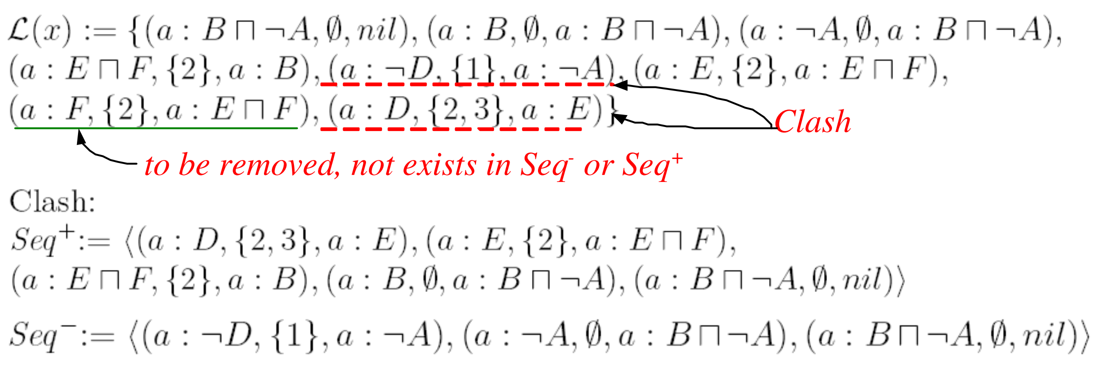
1. If an axiom α_i going to be changed does not exist in the index-set of any sequence of the clashes, i.e., $i \notin I_u$, then α_i is not involved in the unsatisfiability, any change of α_i does not affect the subsumption.

2. If α_i is involved in the unsatisfiability (i.e., $i \in I_u$), but $(- : C, I, -)$ and $(- : C', I, -)$, where $i \in I$, do not appear in any sequences of clashes, then they are not responsible for the clashes in the tree, and therefore not responsible for the subsumption. Since B is negated for the satisfiability test ($A \sqcap \neg B$), we need to check the negated C as well.

∎

We use the following example to illustrate how we make use of the sequences of clashes (satisfiability test between two named concepts) to detect if some change will affect a subsumption.

Example 35 *Given a terminology with the following axioms, we check if $B \sqsubseteq A$.*
α_1: $A \doteq D$
α_2: $B \doteq E \sqcap F$
α_3: $E \sqsubseteq D$



$\mathcal{L}(x) := \{(a : B \sqcap \neg A, \emptyset, nil), (a : B, \emptyset, a : B \sqcap \neg A), (a : \neg A, \emptyset, a : B \sqcap \neg A),$
$(a : E \sqcap F, \{2\}, a : B), (a : \neg D, \{1\}, a : \neg A), (a : E, \{2\}, a : E \sqcap F),$
$(a : F, \{2\}, a : E \sqcap F), (a : D, \{2, 3\}, a : E)\}$

Clash:
$Seq^+ := \langle (a : D, \{2, 3\}, a : E), (a : E, \{2\}, a : E \sqcap F),$
$(a : E \sqcap F, \{2\}, a : B), (a : B, \emptyset, a : B \sqcap \neg A), (a : B \sqcap \neg A, \emptyset, nil)\rangle$
$Seq^- := \langle (a : \neg D, \{1\}, a : \neg A), (a : \neg A, \emptyset, a : B \sqcap \neg A), (a : B \sqcap \neg A, \emptyset, nil)\rangle$

Figure 7.4: Subsumption test on $B \sqsubseteq A$

As seen in Figure 7.4, $B \sqsubseteq A$ holds, because a clash exists in the label of the root node, and then $B \sqcap \neg A$ is unsatisfiable. Assume the concept F in α_2 is to be removed, we know that $B \sqsubseteq A$ still holds, because $(a : F, \{2\}, -)$ does not exist in either of the sequences of the clash.

7.3 Harmful and Helpful Changes

In this section we study ways of changing problematic axioms to resolve unsatisfiability. It should be noted that improperly rewriting a problematic axiom might not resolve the unsatisfiability, and could introduce additional unsatisfiability. It is important to help ontology modellers to make changes in order not to introduce unintended contradictions. For this purpose, we define *harmful* and *helpful* changes. Harmful changes either fail to resolve the existing unsatisfiability or introduce additional unsatisfiability. Helpful changes resolve the problem without causing additional contradictions, and restore some lost entailments.

7.3.1 Harmful Changes

Given an unsatisfiable named concept A in $\mathcal{T}$, assume a concept E on the right-hand side of a problematic axiom α_i is chosen to be replaced by some other concept. We can find the harmful concepts for the replacement of E by analysing the elements in the sequences of the clashes of concept A.

Definition 16 (Harmful Change) *A change which transforms $\mathcal{T}$ to $\mathcal{T}'$ is harmful with respect to an unsatisfiable concept A in $\mathcal{T}$, if one of the following conditions holds:*

- *$\mathcal{T}' \models A \sqsubseteq \bot$, where $\mathcal{T}'$ is the changed ontology;*
- *if some named concept A_i which is satisfiable in $\mathcal{T}$ is not satisfiable in $\mathcal{T}'$. That is, $\mathcal{T} \not\models A_i \sqsubseteq \bot$ and $\mathcal{T}' \models A_i \sqsubseteq \bot$, for some A_i in $\mathcal{T}$.*

The following lemma identifies the changes which are harmful due to the fact that they fail to resolve the existing unsatisfiability. To identify other harmful changes (which introduce additional unsatisfiability unrelated to the original problem), the whole ontology may have to be rechecked.

Lemma 4 *Assume a concept C on the right-hand side of α_i is to be rewritten. Given two sequences of a clash, Seq^+ and Seq^-, involving C, if one of the elements, ε, in Seq^+, is of the form $(a : C, I, -)$ and $i \in I$, then*

1. *All the concepts in the elements in Seq^+ preceding $(a : C, I, -)$, which contain the same individual as ε, are harmful for replacing C;*
2. *The negation of all the concepts in elements in Seq^-, which contain the same individual as ε, are also harmful, because these replacements still keep the unsatisfiability.*

The lemma is analogous for the element ε in Seq^-.

Proof Assume that a concept C on the right-hand side of axiom α_i is to be rewritten, and two sequences of a clash, Seq^+ and Seq^-, involving C, are obtained from a node of the tree **T**. In a sequence, for every two adjacent elements, ε_1 and ε_2, which are of the form $(a : E_1, -, a : E_2)$ and $(a : E_2, -, -)$, containing the same individual, the concept E_1 is a superconcept of E_2. This extends inductively to all elements preceding ε_1, i.e., they are all superconcepts of E_2.

1. If an element ε in Seq^+, which is of the form $(a : C, I, -)$ and $i \in I$, then the concepts in all the elements, which are preceding ε and contain individual a, are harmful for replacing C. This is because they are superconcepts of C which are involved in the clash.
2. The elements in Seq^- lead to a negated concept, which results in a contradiction. Hence, the negation of all the concepts in elements in Seq^-, which contain the same individual a, are also harmful.

∎

Continuing with Example 34 (shown in Figure 7.1) shown as follows:
α_1: $A \doteq C \sqcap \forall R.B \sqcap D$
α_2: $C \doteq \exists R.\neg B \sqcap B$
α_3: $G \doteq \forall R.(C \sqcap F)$
If C in axiom α_1 is going to be replaced, we know that there exists an element $(a : C, \{1\}, a : C \sqcap \forall R.B)$ in Seq^- of the clash, then the harmful concepts for the replacements will be $\exists R.\neg B \sqcap B$, $\exists R.\neg B$, $\neg(\forall R.B)$, $\neg(C \sqcap \forall R.B)$, $\neg(C \sqcap \forall R.B \sqcap D)$, and $\neg A$. The first two items are from Seq^-, the rest are from negated elements in Seq^+.

7.3.2 Helpful Changes

If we know which concepts are harmful to replace a concept in a problematic axiom, then all the concepts which are not harmful are candidates for replacement. However, there are many possible candidates. Our aim is to find desirable concepts for replacement in order to minimise the impact of changes. To do this we introduce *helpful* changes which cover for the lost entailments due to the removal. When an axiom $A \sqsubseteq C$ in $\mathcal{T}$ is changed to be $A \sqsubseteq C'$ (where A is a named concept), this change is helpful if (1) C' can compensate for at least one lost entailment due to the removal of C, (2) the changes are not harmful, that means all concepts which are satisfiable in $\mathcal{T}$ are also satisfiable in the changed ontology. Note that we only change concepts on the right-hand side of axioms. We now formally define a helpful change.

Definition 17 (Helpful Change) *A helpful change is defined as the removal of an axiom followed by an addition. Assume that $\mathcal{T}$ is the original ontology and an axiom α in $\mathcal{T}$ involved in the unsatisfiability of concept A is going to be removed, resulting in intermediate ontology $\mathcal{T}_1$. A new axiom is then added to $\mathcal{T}_1$, resulting in the changed ontology $\mathcal{T}'$. The change is helpful with respect to A, if the following conditions hold:*

1. *if Ω is the set of lost entailments in going from $\mathcal{T}$ to $\mathcal{T}_1$ (i.e., due to the removal of α), such that $\forall\gamma \in \Omega, \mathcal{T} \models \gamma$, then there exists $\beta \in \Omega$, such that $\mathcal{T}_1 \not\models \beta$ and $\mathcal{T}' \models \beta$;*
2. *$\mathcal{T}' \not\models A \sqsubseteq \bot$.*

Lemma 5 *Assume C on the right-hand side of a problematic axiom (involved in the unsatisfiability of A) is going to be replaced by C', the change is helpful if C' is a superconcept of C and is not involved in the clash of A.*

Proof It is obvious that any concept which is not involved in the clash is not harmful as a replacement for C. We now prove its superconcepts are helpful. Given that in an axiom $\alpha : E \sqsubseteq C$ in $\mathcal{T}$, concept C is going to be replaced by its superconcept C'. We divide the change into two steps:

1. Remove C from α, the changed ontology $\mathcal{T}_1 = \mathcal{T} \setminus \{E \sqsubseteq C\}$;
2. Add C' to α, the final ontology $\mathcal{T}' = \mathcal{T}_1 \cup \{E \sqsubseteq C'\}$.

As C' is a superconcept of C, $E \sqsubseteq C$ is removed in $\mathcal{T}_1$, so the indirect subsumption relationships of A with C's superconcepts are also lost, that means $\mathcal{T}_1 \not\models E \sqsubseteq C'$, but obviously, $\mathcal{T}' \models E \sqsubseteq C'$.

∎

Lemma 6 *Given two sequences of a clash w.r.t. the unsatisfiability of A obtained from a fully expanded tree* **T**, *assume a concept C on the right-hand side of axiom α_i is to be rewritten. C' is helpful as a replacement for C, if the following conditions hold:*

1. *There exist two elements e and e' in* **T**, *which are $(a : C, I, -)$ and $(a : C', I', -)$, and no element of the form $(a : C', I', -)$ exists in either of the two sequences;*
2. *$I \subset I'$, the index-set of the element with concept C is a proper subset of that of the element with concept C'.*

Proof We have to prove that (1) C' is a superconcept of C, this is a sufficient condition to ensure that the first requirement for helpfulness is met; and (2) C' is not involved in the clash.

1. If elements e and e' in **T** contain the same individual, then they have a subsumption relationship (i.e., C is either a subconcept or superconcept of C'). Additionally, if the index-set of the element e is a proper subset of the index-set of the element e', then that means e' is added to $\mathcal{L}(x)$ after the addition of e (i.e., the addition of e' is triggered by e). Then we can confirm that C' is a superconcept of C.
2. If an element, which is of the form $(a : C', -, -)$, exists in **T**, but not in either of the two sequences, then C' is not involved in the clash.

∎

Continuing with Example 34 reproduced as follows:
α_1: $A \doteq C \sqcap \forall R.B \sqcap D$
α_2: $C \doteq \exists R.\neg B \sqcap B$
α_3: $G \doteq \forall R.(C \sqcap F)$

We assume that C in axiom α_1 is going to be replaced, there exists an element $(a : C, \{1\}, a : C \sqcap \forall R.B)$ in Seq^- of the clash (see Figure 7.1), we find that the two elements $(a : D, \{1\}, \cdots)$ and $(a : B, \{1, 2\}, \cdots)$ do not exist in either of the sequences of the clash. However, the former element does not fulfill condition (2) in Lemma 6, because the index set of $(a : C, \{1\}, \cdots)$ is not a proper subset of $(a : D, \{1\}, \cdots)$. Hence, the only helpful concept for the replacement is the concept of the latter element, B, as B is a superconcept of A, but D is not.

Overall, the helpful changes include the replacements of a concept by its superconcepts not involved in any clash (see Lemma 6), and the lost entailments irrelevant to the unsatisfiability of the ontology (see Section 7.2). These changes are suggested to the user to add back to the ontology in order to minimise the impact of changes.

7.4 Caching Techniques

In the above section, Lemmas 4 and 6 identify helpful and harmful changes with respect to an unsatisfiable concept. To know if a change is harmful to other satisfiable named concepts, or helpful to other unsatisfiable named concepts, we have to re-check the consistency of the whole ontology by running the tableau algorithm.

Actually, we can just simply check those concepts whose (un)satisfiability involves the axiom to be updated. For example, if an axiom which is going to be rewritten exists in the MUPSs of unsatisfiable concepts, then only the (un)satisfiability of these concepts has to be re-checked. Also the subsumption relations between two named concepts can be reduced to a satisfiability test, we can just re-check those subsumption relations in which the axiom is involved. Parsia et al. [121] use these techniques to optimise the classification and realisation in an ontology.

Before describing our technique to further optimise the satisfiability test of an ontology by *caching*. We now revisit the fine-grained approach described in Chapter 4. To test the satisfiability of concept C, our algorithm initialises a tree **T** to contain single node x, and then expands the node by applying the expansion rules as many times as possible. Whenever a rule is applied, new concept elements or role elements are added to the label of the node. Two leaf nodes (i.e., $\sqcup$-successors) are created and added to the tree whenever the $\sqcup$-rule is applied. In fact, the labels of all leaf nodes in **T** are the ABoxes of C. When the algorithm terminates, we can obtain a finite set of ABoxes of C, $\mathcal{S} = \{\mathcal{A}_1, \cdots, \mathcal{A}_k\}$ from the labels of all leaf nodes. For simplicity of presentation, in the this section, we use the ABoxes of a concept instead of the labels of the leaf nodes in the tree.

When the (un)satisfiability of concepts and subsumption relations is firstly tested by the tableau algorithm, their ABoxes created by the tableau algorithm are cached, so that we can just modify the cached ABoxes to obtain new ABoxes during rewriting axioms, instead of running the tests again. For example, given an axiom ($B \sqsubseteq Z$) which is involved in the (un)satisfiability of a named concept A, when the axiom is modified, we run the tableau algorithm on B to obtain its new ABoxes. With the modified ABoxes of B and the original ABoxes of A, we are able to calculate the modified ABoxes of A without running the tableau algorithm from scratch.

7.4.1 Examples of the Approach

We use a variety of examples to illustrate how this approach works, and then give a generic procedure with the proof of correctness in the next section. Note that the third parameter of concept elements is not shown for simplicity.

7.4.1.1 Adding Union Concept

Example 36 *Given an ontology with two axioms:*

1. $A \sqsubseteq B \sqcap D \sqcap E$

2. $B \sqsubseteq C$

We run the tableau algorithm on A to check its satisfiability and cache its ABoxes, the ABox of A is denoted as $\mathcal{A}_A$.

$$\mathcal{A}_A = \{(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}), (a : B, \{1\}), (a : D, \{1\}), (a : C, \{1, 2\})\}$$

Axiom 2 is changed to $B \sqsubseteq \neg D \sqcup \neg E$, and B's new ABoxes are:

$$\mathcal{A}_{B1} = \{(b : B, \{\}), (b : \neg D \sqcup \neg E, \{2\}), (b : \neg D, \{2\})\}$$
$$\mathcal{A}_{B2} = \{(b : B, \{\}), (b : \neg D \sqcup \neg E, \{2\}), (b : \neg E, \{2\})\}$$

The change of axiom 2 might affect the satisfiability (or ABoxes) of A, so we can run the tableau algorithm to obtain its new ABoxes. However, we observe that it is possible to update the ABoxes of A based on the modified ABoxes of B instead of running the algorithm again. The elements in $\mathcal{A}_A$ which are derived from axiom 2 should be modified when axiom 2 is changed.

$$\mathcal{A}_A = \{(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}), (a : B, \{1\}), (a : D, \{1\}), (\underline{a : C, \{1, 2\}})\} \text{ derived from axiom 2}$$

In our approach, we remove these elements in $\mathcal{A}_A$ and then add the elements from the ABoxes of B except for the element with empty index-set (i.e., $(b : B, \{\})$ in our example). The modified ABoxes of A become:

$$\mathcal{A}_{A1} = \{(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}), (a : B, \{1\}), (a : D, \{1\}), \underbrace{(a : \neg D \sqcup \neg E, \{1, 2\}), (a : \neg D, \{1, 2\})}_{\text{elements from } \mathcal{A}_{B1}}\}$$
$$\mathcal{A}_{A2} = \{(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}), (a : B, \{1\}), (a : D, \{1\}), \underbrace{(a : \neg D \sqcup \neg E, \{1, 2\}), (a : \neg E, \{1, 2\})}_{\text{elements from } \mathcal{A}_{B2}}\}$$

We can see that the index sets of the added elements also got updated with $\{1\}$; the index sets need to keep track of the axioms which each element is derived from. Furthermore, we can see that the individual in the newly added elements is a, even though b was the individual in $\mathcal{A}_B$; we call this a substitution $b \to a$. We know that $\{1\}$ needed to be added and that $b \to a$ was the appropriate substitution by following this method: find the element in $\mathcal{A}_A$ whose concept is the same as the element with empty index-set in the ABoxes of B, and record its index-set and its individual. We call this element a 'matched' element. In this example, the element $(a : B, \{1\}) \in \mathcal{A}_A$ matches the element $(b : B, \{\}) \in \mathcal{A}_B$. Therefore, we add the index-set of the element $(a : B, \{1\}) \in \mathcal{A}_A$ into the index-set of the newly added elements, and the individual b of the newly added elements is substituted by a (written as $b \to a$).

It is obvious that both ABoxes contain a clash because $\{(a : D, \{1\}), (a : \neg D, \{1, 2\})\} \subseteq \mathcal{A}_{A1}$ and $\{(a : E, \{1\}), (a : \neg E, \{1, 2\})\} \subseteq \mathcal{A}_{A2}$ respectively. Note that the axioms in the union of the index-sets of the elements involved in clashes are responsible for the concept's unsatisfiability.

7.4.1.2 Removing Union Concept

Example 37 *The above example shows that a change creates additional ABoxes if a union concept is introduced. We continue this example but make a different change, removing the union concept, so that the number of modified ABoxes is less than the original one. Let us change axiom 2 to be* $B \sqsubseteq \exists R.E$.

The modified ABox of B is:
$\mathcal{A}'_B = \{(b : B, \{\}), (b : \exists R.E, \{2\}), (R(b, c), \{2\}), (c : E, \{2\})\}$

Similar to the above example, we remove the elements in $\mathcal{A}_{A1}$ and $\mathcal{A}_{A2}$ derived from axiom 2, and obtain the following ABoxes:

$$\begin{aligned}\mathcal{A}_{A1} = \{&(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}),\\ &(a : B, \{1\}), (a : D, \{1\}), \underbrace{(a : \neg D \sqcup \neg E, \{1, 2\}), (a : \neg D, \{1, 2\})}_{\text{Derived from axiom 2 => to be removed}}\}\end{aligned}$$

$$\begin{aligned}\mathcal{A}_{A2} = \{&(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}),\\ &(a : B, \{1\}), (a : D, \{1\}), \underbrace{(a : \neg D \sqcup \neg E, \{1, 2\}), (a : \neg E, \{1, 2\})}_{\text{Derived from axiom 2 => to beremoved}}\}\end{aligned}$$

The two ABoxes become the same, and one of them is eliminated – say $\mathcal{A}_{A2}$.

The next step is to find 'matched' elements in the ABoxes of A. The matched element is $(a : B, \{1\})$ in $\mathcal{A}_{A1}$. The elements in $\mathcal{A}'_B$ (except for the one with an empty index-set) are then added to $\mathcal{A}_{A1}$; their index-sets are augmented with $\{1\}$; their individuals are substituted by a (i.e., $b \rightarrow a$). The ABoxes now become:

$$\begin{aligned}\mathcal{A}'_{A1} = \{&(a : A, \{\}), (a : B \sqcap D \sqcap E, \{1\}), (a : B \sqcap D, \{1\}), (a : E, \{1\}),\\ &(a : B, \{1\}), (a : D, \{1\}), \underbrace{(a : \exists R.E, \{1, 2\}), (R(a, c), \{1, 2\}), (c : E, \{1, 2\})}_{\text{elements from } \mathcal{A}'_B}\}\end{aligned}$$

There are no clashes in the ABox $\mathcal{A}'_{A1}$, and hence A is satisfiable.

7.4.1.3 Adding Quantified Restrictions

Example 38 *We now use another example to show how to modify ABoxes when existential or universal restrictions in axioms are changed. Let us assume an ontology with the following two axioms:*

1. $A \sqsubseteq \forall R.B \sqcap C$
2. $C \sqsubseteq \neg B$

By applying the tableau algorithm, the ABox of A is obtained:

$\mathcal{A}_A = \{(a : A, \{\}), (a : \forall R.B \sqcap C, \{1\}), (a : \forall R.B, \{1\}), (a : C, \{1\}), (a : \neg B, \{1, 2\})\}$

Axiom 2 is changed to be $C \sqsubseteq \exists R.\neg B$. The updated ABox of C is
$\mathcal{A}'_C = \{(c : C, \{\}), (c : \exists R.\neg B, \{2\}), (R(c, d), \{2\}), (d : \neg B, \{2\})\}$

The elements in $\mathcal{A}_A$ which are derived from axiom 2 are removed. The elements in $\mathcal{A}'_C$ are added. The 'matched' element is $(a : C, \{1\})$; the index-sets of the newly added elements are augmented with $\{1\}$; the individual c is substituted by a (i.e., $c \rightarrow a$). Therefore, the modified ABox of A is

$$\begin{aligned}\mathcal{A}'_A = \{&(a : A, \{\}), (a : \forall R.B \sqcap C, \{1\}), (a : \forall R.B, \{1\}), (a : C, \{1\}),\\ &\underbrace{(a : \exists R.\neg B, \{1, 2\}), (R(a, d), \{1, 2\}), (d : \neg B, \{1, 2\})}_{\text{elements from } \mathcal{A}'_C}\}\end{aligned}$$

Whenever a relation element is newly added, we have to check if the tableau rules are applicable to the ABoxes. In this example, as the relation element $(R(a,d),\{1,2\})$ is newly added into $\mathcal{A}'_A$, it interacts with $(a:\forall R.B,\{1\})$ to result in more elements to be added in the ABoxes. We apply the $\forall$-rule to $(a:\forall R.B,\{1\})$, and a new element $(d:B,\{1,2\})$ is added to the ABox of A, that is,

$$\begin{aligned}\mathcal{A}'_A = \{&(a:A,\{\}),(a:\forall R.B\sqcap C,\{1\}),(a:\forall R.B,\{1\}),(a:C,\{1\}),(a:\exists R.\neg B,\{1,2\}),\\ &(R(a,d),\{1,2\}),(d:\neg B,\{1,2\}),(d:B,\{1,2\})\}\end{aligned}$$

The ABox of A has a clash because $\{(d:\neg B,\{1,2\}),(d:B,\{1,2\})\}\subseteq\mathcal{A}'_A$, and A becomes unsatisfiable.

7.4.1.4 Multiple Matchings of ABoxes

Example 39 *Sometimes the elements in an ABox also exist in another ABox more than once with the substitution of individuals. We repeat the procedure of updating ABoxes for each substitution of individual. Let us assume an ontology which contains the following two axioms:*

1. $A\sqsubseteq\exists R.B\sqcap B$

2. $B\sqsubseteq C\sqcap D$

The ABox of A is

$$\begin{aligned}\mathcal{A}_A = \{&(a:A,\{\}),(a:\exists R.B\sqcap B,\{1\}),(a:\exists R.B,\{1\}),(a:B,\{1\}),(a:C\sqcap D,\{1,2\}),\\ &(a:C,\{1,2\}),(a:D,\{1,2\}),(R(a,b),\{1\}),(b:B,\{1\}),(b:C\sqcap D,\{1,2\}),\\ &(b:C,\{1,2\}),(b:D,\{1,2\})\}\end{aligned}$$

Assume axiom 2 is now changed to be $B\sqsubseteq E$. The new ABox of B is

$\mathcal{A}'_B=\{(c:B,\{\}),(c:E,\{2\})\}$

In the following, we can see that two sets of elements in $\mathcal{A}_A$ are derived from axiom 2; these elements have to be removed.

$$\begin{aligned}\mathcal{A}_A = \{&(a:A,\{\}),(a:\exists R.B\sqcap B,\{1\}),(a:\exists R.B,\{1\}),(a:B,\{1\}),\\ &\underline{(a:C\sqcap D,\{1,2\}),(a:C,\{1,2\}),(a:D,\{1,2\})},(R(a,b),\{1\}),\\ &(b:B,\{1\}),\underline{(b:C\sqcap D,\{1,2\}),(b:C,\{1,2\}),(b:D,\{1,2\})}\}\end{aligned}$$

derived from axiom 2

The 'matched' elements are $(a:B,\{1\})$ and $(b:B,\{1\})$ in $\mathcal{A}_A$. Next, we add the elements from $\mathcal{A}'_B$ (except for the one with an empty index-set) into $\mathcal{A}_A$, the individuals and index-sets of the newly added elements are augmented with $\{1\}$ and substituted by a (i.e., $c\rightarrow a$). The adding process is repeated except the individuals are substituted by b (i.e., $c\rightarrow b$). Finally, the updated ABox of A becomes:

$$\begin{aligned}\mathcal{A}'_A = \{&(a:A,\{\}),(a:\exists R.B\sqcap B,\{1\}),(a:\exists R.B,\{1\}),(a:B,\{1\}),\\ &\underbrace{(a:E,\{1,2\})}_{\text{element form } \mathcal{A}'_B},(R(a,b),\{1\}),(b:B,\{1\}),\underbrace{(b:E,\{1,2\})}_{\text{element from } \mathcal{A}'_B}\}\end{aligned}$$

7.4.2 Generic Procedure

The algorithm of Table 7.1 describes the generic procedure of the above operations. We explain the algorithm as follows.

1. Given an axiom $i : B \sqsubseteq Z$ involved in the (un)satisfiability of A, we assume the ABoxes of A are cached during the first time of tableau algorithm checking, and are represented by $\mathcal{S}_A$ (this is an input to the algorithm).

2. When axiom i is changed, we run the tableau algorithm on B and cache the modified ABoxes of B, represented by $\mathcal{S}_B$ (this is also an input to the algorithm).

3. For each ABox $\mathcal{A}_A \in \mathcal{S}_A$, the elements whose index-set contain i are removed (cf. line 1-3 in Table 7.1). If some ABoxes become the same, they will be eliminated.

4. We find the elements in $\mathcal{A}_A$ of the form $(- : B, -)$, and store them into $matchSet$ (cf. line 7-8).

5. For each element $(a : B, I)$ in the $matchSet$, we do the following: for each ABox of B, $\mathcal{A}_B \in \mathcal{S}_B$, all elements (except the one with empty index-set), are added to $\mathcal{A}_A$. The index-set of newly added elements is augmented with I; their individual is also substituted by a. (cf. line 9-15)

6. After updating each ABox $\mathcal{A}_A$, we have to check if the expansion rules are applicable to the ABox. If so, the ABoxes $\mathcal{S}_A$ are updated by applying the expansion rules.

7. The ABoxes $\mathcal{S}_A$ are checked for any clash, and hence we know if A is satisfiable after the change.

Table 7.1: Algorithm for updating ABoxes

```
     Input: A modified axiom $i$: $B \sqsubseteq Z$, where $B$ is a named concept,
     $\mathcal{S}_A$ is the ABoxes of $A$ , $\mathcal{S}_B$ is the modified ABoxes of $B$
     Procedure: UpdateABoxes($\mathcal{S}_A, \mathcal{S}_B, i, B$)
1.   for each $\mathcal{A}_A \in \mathcal{S}_A$
2.     for each $(a : D, I) \in \mathcal{A}_A$, where $D$ is any concept description
3.       if $(i \in I)$ then $\mathcal{A}_A := \mathcal{A}_A \setminus \{(a : D, I)\}$
4.   eliminate_duplicated($\mathcal{S}_A$);
5.   for each $\mathcal{A}_A \in \mathcal{S}_A$
6.     $matchSet := \{\}$;
7.     for each $(a : B, I) \in \mathcal{A}_A$
8.       $matchSet := matchSet \cup \{(a : B, I)\}$
9.     for each $(a : B, I) \in matchSet$
10.      for each $\mathcal{A}_B \in \mathcal{S}_B$
11.        $\mathcal{A}'_A := \mathcal{A}_A$
12.        $\mathcal{A}_B := \mathcal{A}_B \setminus \{(b : B, \{\})\}$
13.        for each $(d : E, I') \in \mathcal{A}_B$, where $E$ is any concept description
14.          if $(b = d)$, then $\mathcal{A}'_A := \mathcal{A}'_A \cup \{(a : E, I \cup I')\}$
15.          else $\mathcal{A}'_A := \mathcal{A}'_A \cup \{(d : E, I \cup I')\}$
16.        $\mathcal{S}_A := \mathcal{S}_A \setminus \{\mathcal{A}_A\} \cup \{\mathcal{A}'_A\}$
17.  return $\mathcal{S}_A$
     ApplyTableauRules($\mathcal{S}_A$)
```

7.4.3 Proof of Correctness

To prove the correctness of UpdateABoxes, we only need to prove that it has the same effect as running the tableau algorithm on the modified ontology, from scratch. That is, we have to prove that the set of ABoxes generated by our procedure is the same as the set of ABoxes produced by running the tableau from scratch. For the purpose of comparison we show the tableau rules of Table 4.1 (except for the $\sqsubseteq$-rule) as algorithm Tableau in Table 7.2. To facilitate the proof, rather than directly proving the equivalence of Tableau and UpdateABoxes, we find it convenient to introduce an intermediate UpdateABoxesModified (see Table 7.3). Similar to UpdateABoxes, the new version UpdateABoxesModified also adds elements to the ABoxes of A by copying them from a separate copy of the ABoxes of B; however, UpdateABoxesModified builds its separate copy of the ABoxes of B itself and adds elements to the ABoxes of A during each building step. This means that UpdateABoxesModified can mirror the building process of Tableau. We first prove that UpdateABoxesModified produces the same results as Tableau; in this we need to prove that the substitution of individuals and updates to index-sets ensure that the elements which UpdateABoxesModified adds to ABoxes of A are the same as those added by Tableau. Finally, we prove UpdateABoxes and UpdateABoxesModified produce the same results; this second part is more straightforward to see as both are working on the same principle of adding elements to ABoxes of A by copying them from a separate copy of the ABoxes of B.

```
     Input: a concept $A$ in an ontology $\mathcal{O}$
     procedure: Tableau($A, \mathcal{O}$)
1.   $\mathcal{A}^t_A := \{(a : A, \{\})\}$;   $\mathcal{S}^t_A := \{\mathcal{A}^t_A\}$;
2.   for each $\mathcal{A}^t_A \in \mathcal{S}^t_A$
3.      repeat
4.         Try $U^+_{\dot{=}}$-rule; (see Table 4.1)
5.         Try $U^-_{\dot{=}}$-rule;
6.         Try $U_{\sqsubseteq}$-rule;
7.         Try $\sqcap$-rule;
8.         Try $\exists$-rule;
9.         Try $\forall$-rule;
10.        Try $\sqcup$-rule; if $\sqcup$-rule succeeds then $\mathcal{S}^t_A := \mathcal{S}^t_A \cup \mathcal{A}'_A$
11.     until no Tableau rule is applicable;
12.  end for
13.  return $\mathcal{S}^t_A$;
```

Table 7.2: Tableau Algorithm to Test Satisfiability of A

Lemma 7 *Given: (i) an ontology $\mathcal{O}$, and a modified version $\mathcal{O}'$ which differ only in axiom i; (ii) a named concept B, which is defined by axiom i of the ontologies ($i : B \sqsubseteq Z$); (iii) a concept A whose definition depends on concept B; (iv) the set of original ABoxes of A is $\mathcal{S}_A$, i.e., $\mathcal{S}_A =$*Tableau*$(A,\mathcal{O})$.*
We claim: The procedure UpdateABoxesModified*($\mathcal{S}_A, i, B, \mathcal{O}'$) returns a set of new ABoxes of A which is the same as the set of ABoxes produced by* Tableau$(A, \mathcal{O}')$.

Proof UpdateABoxesModified works by incrementally building up the ABoxes $\mathcal{S}_A$, while Tableau incrementally builds up the ABoxes $\mathcal{S}^t_A$; we need to show that when both finish $\mathcal{S}_A = \mathcal{S}^t_A$.

We have created UpdateABoxesModified so that it builds up $\mathcal{S}_A$ in exactly the same way as Tableau. Thus we will be able to prove that their intermediate constructions match each other at each step. Our proof is inductive. We first show how UpdateABoxesModified deletes the elements of $\mathcal{S}_A$ which related to axiom i and hence starts at the same point as Tableau, before Tableau applies unfolding rules to axiom i. Thereafter we claim that every addition which UpdateABoxesModified makes to $\mathcal{S}_A$ is exactly the same as the next addition which Tableau makes to $\mathcal{S}^t_A$; we do this inductively by simply showing that from any given state of partial construction, each algorithm will apply the same rule, and the addition (to $\mathcal{S}^t_A$ or $\mathcal{S}_A$) will be the same. See Table 7.4 for details.

In the first steps of UpdateABoxesModified (cf. line 1–4 in Table 7.3), Tableau and UpdateABoxesModified will have the same ABoxes of A. Whenever an expansion rule is applied on $\mathcal{A}^t_A$ and $\mathcal{A}_B$, the elements with the same concept are created. In UpdateABoxesModified we add these elements with modified index-set and individual into $\mathcal{A}_A$ (cf. line 12–28 in Table 7.3). If the original ABoxes of A, $\mathcal{S}_A$, contain more than one element of the form $(- : B, -)$, the above process will be repeated (cf. line 9). The same happens in Tableau. Therefore, we can conclude that UpdateABoxesModified($\mathcal{S}_A, i, B$) returns a set of ABoxes which is the same as the set of ABoxes produced by Tableau$(A, \mathcal{O})$.

∎

We now compare UpdateABoxesModified and UpdateABoxes.

Lemma 8 UpdateABoxesModified*($\mathcal{S}_A, i, B, \mathcal{O}'$) returns a set of ABoxes which is the same as the set of ABoxes produced by* UpdateABoxes*($\mathcal{S}_A, \mathcal{S}_B, i, B$).*

Proof (Sketch) Lines 1–4 are the same in both procedures. It should be easy to see that thereafter they both add the same elements to $\mathcal{S}_A$, except that the order of adding is slightly different. In UpdateABoxes, the ABoxes of B, $\mathcal{S}_B$, are already created, each element in the set of ABoxes of B except that one with empty index-set is added into each $\mathcal{A}_A$, which is the ABox of A containing elements of form $(- : B, -)$ (cf. line 9–15 in Table 7.1). The index-set of the newly created elements and individuals are updated (cf. line 14). On the other hand, in UpdateABoxesModified, during the application of expansion rules on $\mathcal{A}_B$, whenever new elements are created, they are added into the $\mathcal{A}_A$ at the same time; their index-sets and individuals are updated (cf. line 12–28 in Table 7.3). If a new ABox of B is created, then a new ABox of A is also created (cf. line 24–28), the newly created ABox of A is then added into $\mathcal{S}_A$ for updating (cf. line 27). The main difference (in order) is here: UpdateABoxesModified will add all the elements (which it had built in the ABox of B) in one go, whereas UpdateABoxes will always add elements one by one.

∎

7.5 Related Work

As described in Chapter 3, several methods have been developed in the literature to deal with unsatisfiable ontologies. In this section, we compare our approach with those calculate the impact of removal on an ontology, as well as use caching techniques to optimise reasoning in updating ontologies.

7.5.1 Impact of Removal

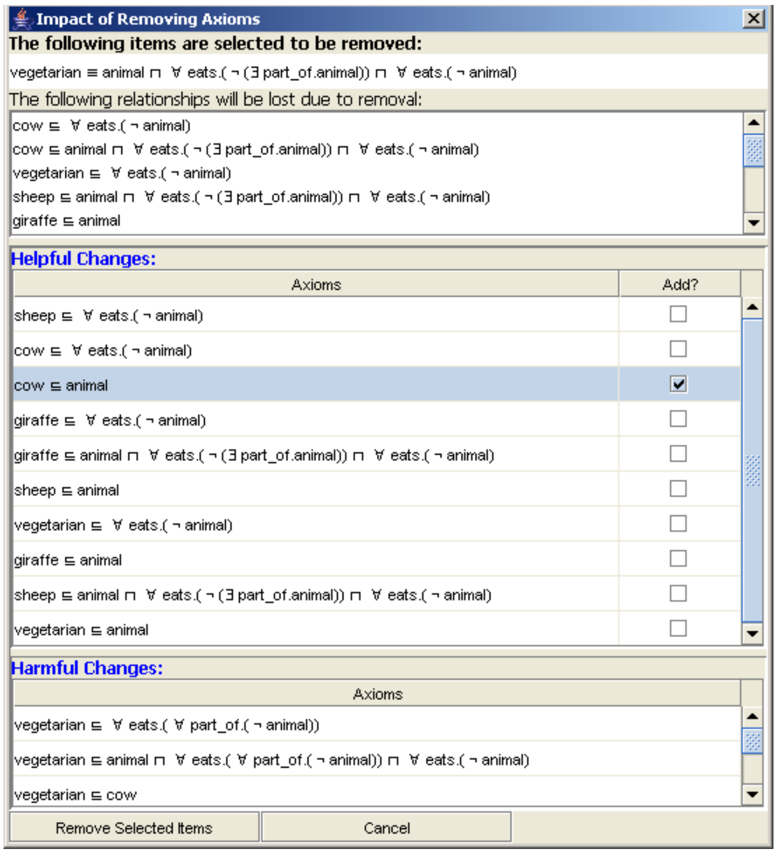
Kalyanpur et al. [91], the authors of SWOOP, analyse the impact on an ontology when an axiom is removed. Currently, they only consider the the subsumption/disjointness between two named concepts (i.e., $A \sqsubseteq B$) and an instantiation (i.e., $B(a)$) which will be lost due to axiom removal. The difference with our work is the following:

1. The lost entailments we consider in Section 7.2, which are not relevant to a concept's unsatisfiability, can be added back to the ontology (i.e., helpful changes), whereas this feature is not available in their approach.

2. We calculate the lost entailments of named concepts when a *part* of an axiom or a whole axiom is removed; they only consider the impact when a whole axiom is removed.

3. We adapt the "difference" operator to calculate the lost entailment of a concept (see Section 7.2.2); their lost entailment is limited to subsumption/disjointness between two named concepts and instantiations.

Continuing our mad_cow example in Example 33, if α_4 is removed, their lost entailment is Cow $\sqsubseteq$ Animal[2]. See Figure 7.5 to compare with our results.



Impact of Removing Axioms

The following items are selected to be removed:

vegetarian ≡ animal ⊓ ∀ eats.(¬ (∃ part_of.animal)) ⊓ ∀ eats.(¬ animal)

The following relationships will be lost due to removal:

cow ⊑ ∀ eats.(¬ animal)
cow ⊑ animal ⊓ ∀ eats.(¬ (∃ part_of.animal)) ⊓ ∀ eats.(¬ animal)
vegetarian ⊑ ∀ eats.(¬ animal)
sheep ⊑ animal ⊓ ∀ eats.(¬ (∃ part_of.animal)) ⊓ ∀ eats.(¬ animal)
giraffe ⊑ animal

Helpful Changes:

Axioms	Add?
sheep ⊑ ∀ eats.(¬ animal)	☐
cow ⊑ ∀ eats.(¬ animal)	☐
cow ⊑ animal	✓
giraffe ⊑ ∀ eats.(¬ animal)	☐
giraffe ⊑ animal ⊓ ∀ eats.(¬ (∃ part_of.animal)) ⊓ ∀ eats.(¬ animal)	☐
sheep ⊑ animal	☐
vegetarian ⊑ ∀ eats.(¬ animal)	☐
giraffe ⊑ animal	☐
sheep ⊑ animal ⊓ ∀ eats.(¬ (∃ part_of.animal)) ⊓ ∀ eats.(¬ animal)	☐
vegetarian ⊑ animal	☐

Harmful Changes:

Axioms
vegetarian ⊑ ∀ eats.(∀ part_of.(¬ animal))
vegetarian ⊑ animal ⊓ ∀ eats.(∀ part_of.(¬ animal)) ⊓ ∀ eats.(¬ animal)
vegetarian ⊑ cow

Figure 7.5: Impact of removing the vegetarian axiom in the mad_cow example

7.5.2 Caching Techniques

There are two most relevant works which use caching techniques to optimise reasoning in updating ontologies. Parsia et al. [121] propose caching techniques to reduce the number of consistency checks required during classification and realisation. During axiom additions, consistency checks for the previously known subsumption relations and concept instantiation are avoided. During axiom deletions, consistency checks for the previously known non-subsumptions and non-instantiations are also avoided. Halaschek-Wiener et al. [66] present an algorithm for updating completion graphs under both the addition and removal of ABox assertions. Their approach is to update a previously constructed tableau completion graph in such a way that it is guaranteed to correspond to a model of the updated ontology. When an assertional axiom is added to the ontology, it is directly added to the completion graph based on the tableau algorithm. When an assertional axiom is removed, the completion graph can be rolled back, this can be achieved by the tracing function [88]. By and large, we have the same aim as their work, that is to optimise the reasoning time for an ontology during updating axioms. However, our work focuses on terminological reasoning instead of assertional; our technique is also applicable for assertional axioms updating.

7.6 Conclusion

In this chapter we have presented methods for calculating the lost entailments of named concepts due to removal of (parts of) axioms. Our technique is also able to identify harmful and helpful changes for concepts which are going to be replaced. With our approach, users are provided with support to help them to rewrite axioms in order to resolve the problems with *minimal* impact on the ontology. Furthermore, a caching technique is proposed to improve the efficiency of consistency checking for modified ontologies. We have conducted a usability evaluation to demonstrate the applicability of our approach in practice. The results are presented in Chapter 8.

```
      Given:
(i)   an ontology $\mathcal{O}$, and a modified version $\mathcal{O}'$ which differ only in axiom $i$;
(ii)  a named concept $B$, which is defined by axiom $i$ of the ontologies ($i : B \sqsubseteq Z$);
(iii) a concept $A$ where the definition of $A$ depends on concept $B$;
(iv)  the set of original ABoxes of $A$ is $\mathcal{S}_A$, i.e., $\mathcal{S}_A =$Tableau$(A,\mathcal{O})$.
      Procedure: UpdateABoxesModified($\mathcal{S}_A, i, B, \mathcal{O}'$)
1.    for each $\mathcal{A}_A \in \mathcal{S}_A$
2.       for each $(a : D, I) \in \mathcal{A}_A$, where $D$ is any concept description
3.          if ($i \in I$) then $\mathcal{A}_A := \mathcal{A}_A \setminus \{(a : D, I)\}$
4.    eliminate_duplicated($\mathcal{S}_A$);
5.    for each $\mathcal{A}_A \in \mathcal{S}_A$
6.       $matchSet := \{\}$;
7.       for each $(a : B, I) \in \mathcal{A}_A$
8.          $matchSet := matchSet \cup \{(a : B, I)\}$
9.       for each $(a : B, I) \in matchSet$
10.         $\mathcal{A}_B := \{(b : B, \{\})\}$;    $\mathcal{S}_B := \{\mathcal{A}_B\}$
11.         loop
               *(note that below we apply the tableau rules to $\mathcal{S}_B$ using ontology $\mathcal{O}'$)*
12.            if $U^+_{\doteq}$-rule, then $\mathcal{A}_B := \mathcal{A}_B \cup \{(b : C, \{i\})\}$
13.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(a : C, I \cup \{i\})\}$
14.            else if $U^-_{\doteq}$-rule, then $\mathcal{A}_B := \mathcal{A}_B \cup \{(b : \neg C, \{i\})\}$
15.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(a : \neg C, I \cup \{i\})\}$
16.            else if $U_{\sqsubseteq}$-rule, then $\mathcal{A}_B := \mathcal{A}_B \cup \{(b : C, \{i\})\}$
17.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(a : C, I \cup \{i\})\}$
18.            else if $\sqcap$-rule, then $\mathcal{A}_B := \mathcal{A}_B \cup \{(b : C_1, \{i\}), (b : C_2, \{i\})\}$
19.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(a : C_1, I \cup \{i\}), (a : C_2, I \cup \{i\})\}$
20.            else if $\exists$-rule, then $\mathcal{A}_B := \mathcal{A}_B \cup \{(R(b, b1), \{i\}), (b1 : C, \{i\})\}$
21.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(R(a, a1), I \cup \{i\}), (a1 : C, I \cup \{i\})\}$
22.            else if $\forall$-rule, then $\mathcal{A}_B := \mathcal{A}_B \cup \{(b1 : C_1, \{i\})\}$
23.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(b1 : C_1, I \cup \{i\})\}$
24.            else if $\sqcup$-rule, then
25.                  $\mathcal{A}'_B := \mathcal{A}_B \cup \{(b : C_1, \{i\}), (b : C_2, \{i\})\}$;
                        $\mathcal{S}_B := \mathcal{S}_B \cup \{\mathcal{A}'_B\}$
26.                  $\mathcal{A}_B := \mathcal{A}_B \cup \{(b : C_2, \{i\})\}$
27.                  $\mathcal{A}'_A := \mathcal{A}_A \cup \{(a : C_1, I \cup \{i\}), (a : C_2, I \cup \{i\})\}$;
                        $\mathcal{S}_A := \mathcal{S}_A \cup \{\mathcal{A}'_A\}$
28.                  $\mathcal{A}_A := \mathcal{A}_A \cup \{(a : C_1, I \cup \{i\}), (a : C_2, I \cup \{i\})\}$
29.            else end loop;
30.   return $\mathcal{S}_A$;
```

Table 7.3: Modified version of UpdateABoxes

	Tableau	UpdateABoxesModified
init.	Initialise $\mathcal{A}^t_A :=$ $\{(a : A, \{\})\}$; $\mathcal{S}^t_A := \{\mathcal{A}^t_A\}$. Apply expansion rules to $(a : A, \{\})$; pause it before the unfolding rule is applied to $(a' : B, I_a)$.	We take the original $\mathcal{S}_A$, and roll back to the status before the application of unfolding rules on the element with concept B. This is done by removing elements whose index-set contains axiom i (cf. Line 1-4 in Table 7.3). The set of ABoxes is the same as $\mathcal{S}^t_A$.
U-rule	Apply unfolding rules to $(a' : B, I_a)$. New element $(a' : Z, I_a \cup \{i\})$ is added to $\mathcal{A}^t_A$. Let's say $I_1 := I_a \cup \{i\}$.	Initialise $\mathcal{A}_B := \{(b : B, \{\})\}$; $\mathcal{S}_B := \{\mathcal{A}_B\}$. Apply unfolding rule to $(b : B, \{\})$. New element $(b : Z, \{i\})$ is added to $\mathcal{A}_B$. We copy the new element to the ABox of A which contains $(a' : B, I_b)$. Let's say that ABox in A is called $\mathcal{A}_A$. The new element is modified as $(a' : Z, I_b \cup \{i\})$. Let's say $I_2 := I_b \cup \{i\}$.
$\sqcap$-rule	Two elements $(a1 : C_1, I_1)$ and $(a1 : C_2, I_1)$ are added to $\mathcal{A}^t_A$.	Two elements $(b1 : C_1, I_b)$ and $(b1 : C_2, I_b)$ are added to $\mathcal{A}_B$. Copy them to $\mathcal{A}_A$. The new elements are modified as $(a' : C_1, I_2)$, $(a' : C_2, I_2)$.
$\exists$-rule	Two elements $(R(a1, a2), I_1)$ and $(a2 : E, I_1)$ are added to $\mathcal{A}^t_A$.	Two elements $(R(b1, b2), I_b)$ and $(b2 : E, I_b)$ are added to $\mathcal{A}_B$. The modified elements $(R(a', b2), I_2)$, $(b2 : E, I_2)$ are added to $\mathcal{A}_A$.
$\forall$-rule	An element $(a2 : F, I_1)$ is added to $\mathcal{A}^t_A$.	An element $(b2 : F, I_b)$ is added to $\mathcal{A}_B$. The modified element $(b2 : F, I_2)$ is added to $\mathcal{A}_A$.
$\sqcup$-rule	A new ABox, $\mathcal{A}^{t1}_A := \mathcal{A}^t_A$, is created. $\mathcal{A}^{t1}_A$ is augmented with $(a1 : G, I_1)$; $\mathcal{A}^t_A$ is augmented with $(a1 : K, I_1)$.	A new ABox, $\mathcal{A}_{B1} := \mathcal{A}_B$, is created. $\mathcal{A}_{B1}$ is added with $(b1 : G, I_b)$; $\mathcal{A}_B$ is added with $(b1 : K, I_b)$. A new ABox of A is created such that $\mathcal{A}'_A := \mathcal{A}_A$. $\mathcal{A}'_A$ is added with $(a' : G, I_2)$; $\mathcal{A}_A$ is added with $(a' : K, I_2)$; $\mathcal{S}_A$ is added with $\mathcal{A}'_A$.

Table 7.4: Procedures of Tableau and UpdateABoxesModified

Chapter 8

Implementation and Evaluation

This chapter describes 'RepairTab', an ontology debugging tool which has been implemented as a test-bed for the methods proposed in Chapters 4–7, namely the heuristic-based service and the fine-grained approach. The practical significance of these methods is demonstrated by a number of empirical studies and by comparing RepairTab with existing debugging tools. Two usability studies were conducted to determine the practical usefulness of (1) the heuristic-based service which ranks problematic axioms, and (2) the fine-grained approach which provides detailed information about the impact of changes on the ontology, as well as identifies changes which are helpful in that they restore lost entailments (due to axiom removal), and those that are harmful in that they cause additional unsatisfiability.

The chapter is organised as follows: Section 8.1 describes the implementation details of two proposed methods. Section 8.2 provides the details and results of the usability study of the heuristic-based service. Section 8.3 illustrates benefits of our fine-grained approach when compared with other debugging tools. Section 8.4 compares the advantages and disadvantages of RepairTab with other debugging tools.

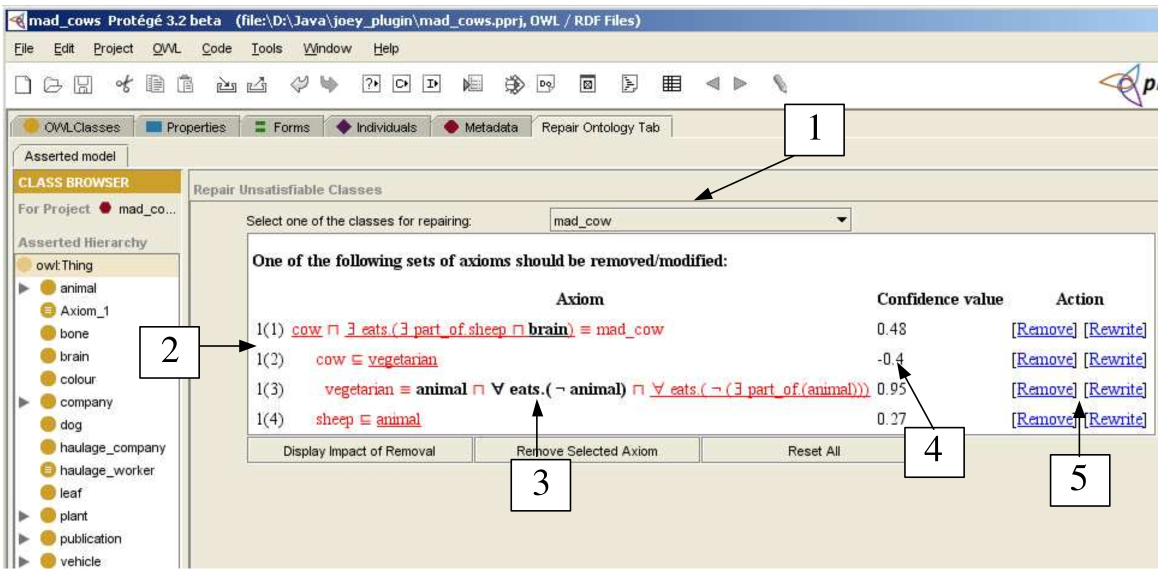
8.1 Implementation

To demonstrate the usefulness of our proposed approaches we carried out two usability studies, the aim of which was to determine how useful the proposed methods are when incorporated to an ontology debugging tool. For this purpose we built a Protégé 3.2[1] plugin, called, "RepairTab". Our implementation extends the Pellet[2] reasoner. Pellet is an open source reasoner which implements the standard tableaux algorithm; it was chosen because it is a highly optimised implementation. We extended it to support our fine-grained and heuristic approaches. In this section we describe the functionalities provided by RepairTab, which is able to identify faulty parts of axioms, rank problematic axioms, as well as minimise the impact of change on the ontology by providing helpful and harmful changes. Figure 8.1 shows the plugin displaying the problematic axioms of mad_cow, which is an unsatisfiable concept from the Mad_Cow ontology[3].

We describe the functionalities of the plugin as follows:

1. A combo box (cf. 1 in Figure 8.1) lists all the unsatisfiable concepts. If there is more than

[1]http://protege.stanford.edu/plugins/owl/
[2]http://www.mindswap.org/2003/pellet/
[3]http://cohse.semanticweb.org/ontologies/people.owl


mad_cows Protégé 3.2 beta (file:\D:\Java\joey_plugin\mad_cows.pprj, OWL / RDF Files)
File Edit Project OWL Code Tools Window Help
OWLClasses Properties Forms Individuals Metadata Repair Ontology Tab
Asserted model
CLASS BROWSER
For Project mad_co...
Asserted Hierarchy
owl:Thing
animal
Axiom_1
bone
brain
colour
company
dog
haulage_company
haulage_worker
leaf
plant
publication
vehicle
Repair Unsatisfiable Classes
Display Impact of Removal
Remove Selected Axiom
Reset All
1
2
3
4
5

Select one of the classes for repairing: mad_cow

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	cow ⊓ ∃ eats.(∃ part_of.sheep ⊓ **brain**) ≡ mad_cow	0.48	[Remove] [Rewrite]
1(2)	cow ⊑ vegetarian	-0.4	[Remove] [Rewrite]
1(3)	vegetarian ≡ **animal** ⊓ **∀ eats.(¬ animal)** ⊓ ∀ eats.(¬(∃ part_of.(animal)))	0.95	[Remove] [Rewrite]
1(4)	sheep ⊑ animal	0.27	[Remove] [Rewrite]

Figure 8.1: Screenshot of the "RepairTab" plugin

one concept, the user is recommended to resolve the concepts from the top of the list.

2. The list of axioms causing the concept to be unsatisfiable is displayed (cf. 2).

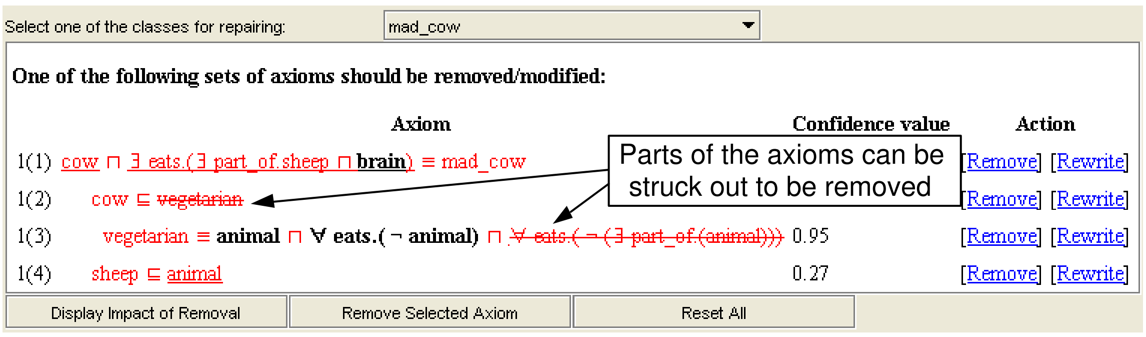
3. The problematic parts of axioms are highlighted. The **BOLD** concept names are not responsible for the unsatisfiability. Concept names not bold are responsible for the unsatisfiability (cf. 3).

4. The confidence value of the axioms is shown. Higher confidence values mean that the axiom is recommended to be preserved; lower confidence values mean that the axiom is recommended to be removed/modified (cf. 4). Figure 8.1 shows that the axiom cow $\sqsubseteq$ vegetarian has a lower confidence value, that means, the system suggests that this axiom is the best for removal/modifiction. The user can follow or ignore the suggestions given by the system to resolve the unsatisfiability.

5. Two types of actions are available: `[Remove]`, `[Rewrite]` (cf. 5). If the user clicks on the link `[Remove]`; the whole axiom will be struck out.

6. The system also allows the user to remove parts of axioms which are responsible for the unsatisfiability. (Figure 8.2 shows that the parts of the axioms 3 and 4 are struck out to be removed.)

7. If the user clicks on `Display Impact of Removal` (in Figure 8.2), the window displaying the `Impact of Removing Axiom` will pop up (see Figure 8.3). It shows which entailments are lost if (part of) the axiom is removed. In our example, if the user decides to remove vegetarian from the axiom cow $\sqsubseteq$ vegetarian, the lost entailments of this removal can be previewed. Figure 8.3 shows that two entailments, cow $\sqsubseteq$ animal and cow $\sqsubseteq$ ∀eats.(¬animal), will be lost. The harmful and helpful changes are also listed. Hence, the user can choose to add the helpful changes back to the ontology to minimise the impact


Parts of the axioms can be struck out to be removed
Display Impact of Removal
Remove Selected Axiom
Reset All

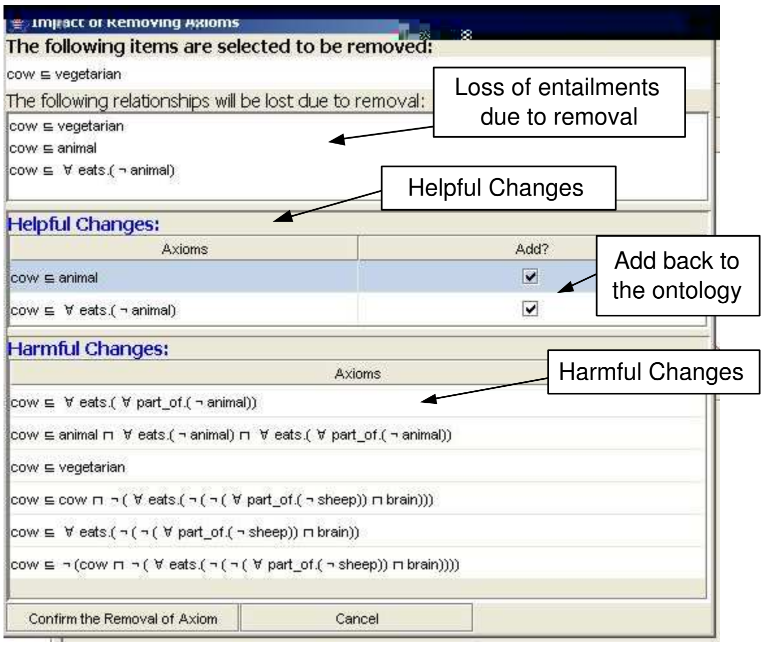
Select one of the classes for repairing: mad_cow

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	cow ⊓ ∃ eats.(∃ part_of.sheep ⊓ **brain**) ≡ mad_cow		[Remove] [Rewrite]
1(2)	cow ⊑ ~~vegetarian~~		[Remove] [Rewrite]
1(3)	vegetarian ≡ **animal** ⊓ **∀ eats.(¬ animal)** ⊓ ~~∀ eats.(¬ (∃ part_of.(animal)))~~	0.95	[Remove] [Rewrite]
1(4)	sheep ⊑ animal	0.27	[Remove] [Rewrite]

Figure 8.2: Parts of the axioms are struck out to be removed

of the removal. As mentioned before, all concepts are normalised in the tableaux algorithm, the generated harmful and helpful changes are hence in negation normal form. That means that the format of displayed changes of axioms might be different from the original ones. To help users understand our suggestions better, further work will involve displaying the information in its original format.


Impact of Removing Axioms
Loss of entailments due to removal
Helpful Changes
Add back to the ontology
Harmful Changes
Confirm the Removal of Axiom
Cancel

The following items are selected to be removed:

cow ⊑ vegetarian

The following relationships will be lost due to removal:

cow ⊑ vegetarian
cow ⊑ animal
cow ⊑ ∀ eats.(¬ animal)

Helpful Changes:

Axioms	Add?
cow ⊑ animal	✓
cow ⊑ ∀ eats.(¬ animal)	✓

Harmful Changes:

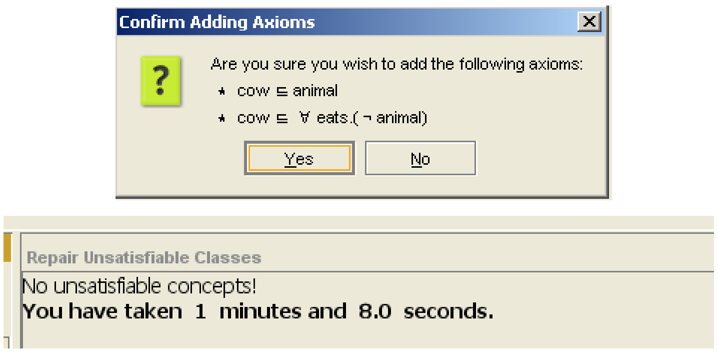
Axioms
cow ⊑ ∀ eats.(∀ part_of.(¬ animal))
cow ⊑ animal ⊓ ∀ eats.(¬ animal) ⊓ ∀ eats.(∀ part_of.(¬ animal))
cow ⊑ vegetarian
cow ⊑ cow ⊓ ¬ (∀ eats.(¬ (¬ (∀ part_of.(¬ sheep)) ⊓ brain)))
cow ⊑ ∀ eats.(¬ (¬ (∀ part_of.(¬ sheep)) ⊓ brain))
cow ⊑ ¬ (cow ⊓ ¬ (∀ eats.(¬ (¬ (∀ part_of.(¬ sheep)) ⊓ brain))))

Figure 8.3: Impact of removal on the ontology

8. The user can click on the button `Confirm the Removal of Axiom` to execute the removal from the ontology, or he can click on the `Cancel` button to cancel the removal.

9. If the user clicks the checkboxes on the list of helpful changes to add them back to the ontology, then he has to confirm this action. The system checks the consistency of the ontology, and shows any unsatisfiable concepts. (see Figure 8.4)

10. On the other hand, if the user clicks on `[Rewrite]`, he can type his own axiom to replace the current one. (Figure 8.5 shows this feature.)



Confirm Adding Axioms

Are you sure you wish to add the following axioms:

- cow ⊑ animal
- cow ⊑ ∀ eats.(¬ animal)

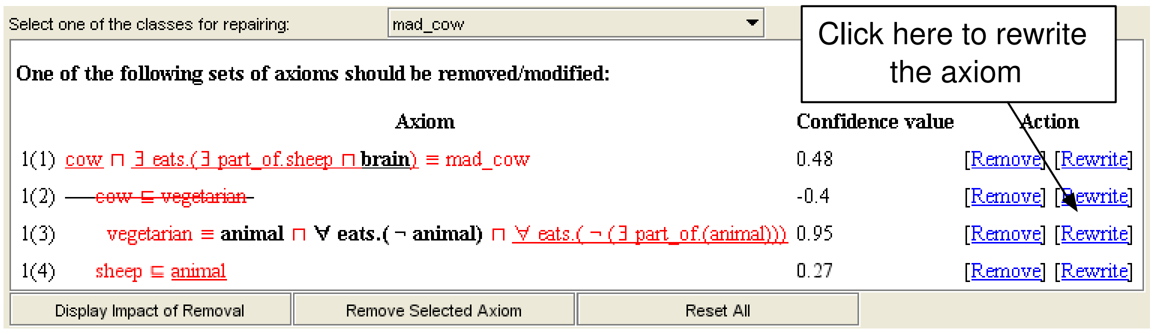
Yes No

Repair Unsatisfiable Classes

No unsatisfiable concepts!

You have taken 1 minutes and 8.0 seconds.

Figure 8.4: Confirmation of adding helpful changes back to the ontology



Select one of the classes for repairing: mad_cow

Click here to rewrite the axiom

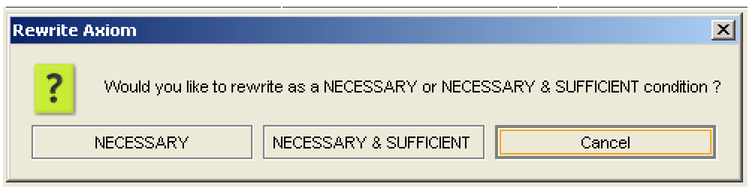
One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	cow ⊓ ∃ eats.(∃ part_of.sheep ⊓ **brain**) ≡ mad_cow	0.48	[Remove] [Rewrite]
1(2)	~~cow ⊑ vegetarian~~	-0.4	[Remove] [Rewrite]
1(3)	vegetarian ≡ **animal** ⊓ **∀ eats.(¬ animal)** ⊓ ∀ eats.(¬ (∃ part_of.(animal)))	0.95	[Remove] [Rewrite]
1(4)	sheep ⊑ animal	0.27	[Remove] [Rewrite]

Display Impact of Removal | Remove Selected Axiom | Reset All

Figure 8.5: Rewriting an Axiom

11. The user has to choose to rewrite the axiom as a ‘Necessary’ or ‘Necessary and Sufficient’ condition. (see Figure 8.6)



Rewrite Axiom

Would you like to rewrite as a NECESSARY or NECESSARY & SUFFICIENT condition ?

NECESSARY | NECESSARY & SUFFICIENT | Cancel

Figure 8.6: Choose a ‘Necessary’ or ‘Necessary and Sufficient’ condition

12. The user can type any concept description to replace the current axiom, and then click `Check the consequences of rewriting`. (see Figure 8.7)

13. If the user clicks the button `Check the consequences of rewriting`, the system will check the consistency of the ontology and display the fixed unsatisfiable concepts, (see Figure 8.8).

14. If the user confirms his rewriting, the system will check the satisfiability of the ontology again (see Figure 8.9).

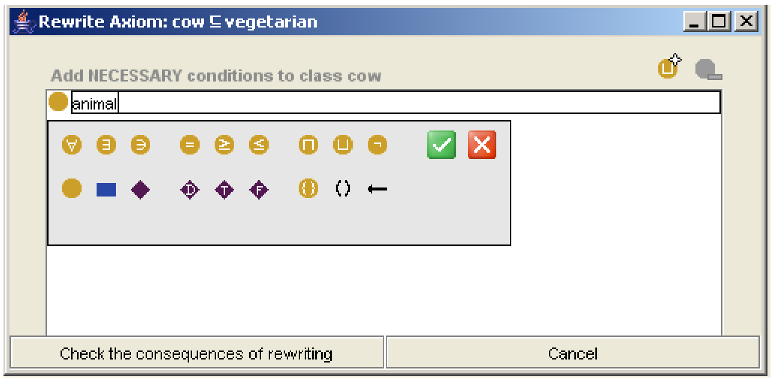

Rewrite Axiom: cow ⊑ vegetarian
Add NECESSARY conditions to class cow
animal
Check the consequences of rewriting
Cancel

Figure 8.7: Rewriting an axiom

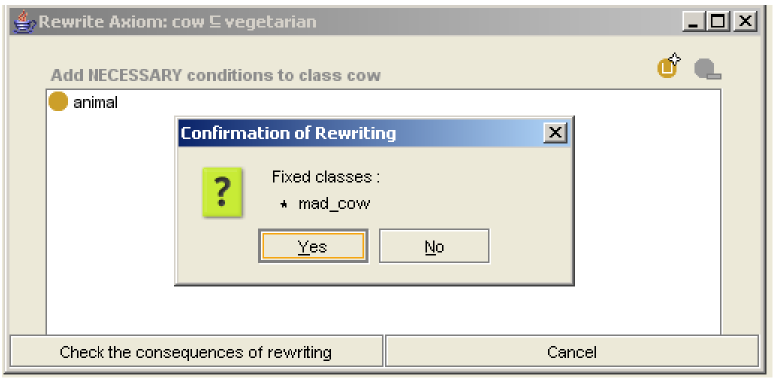
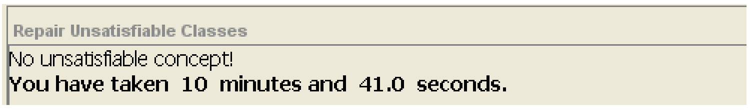

Rewrite Axiom: cow ⊑ vegetarian
Add NECESSARY conditions to class cow
animal
Confirmation of Rewriting
Fixed classes :
mad_cow
Yes
No
Check the consequences of rewriting
Cancel

Figure 8.8: Confirm rewriting an axiom


Repair Unsatisfiable Classes

No unsatisfiable concept!
You have taken 10 minutes and 41.0 seconds.

Figure 8.9: Check the consistency of ontology again

8.2 Evaluation of the Heuristic-based Service

To demonstrate the usefulness of the heuristic-based service we carried out a usability study. The aims of the study were to (1) test if the heuristics can be effectively applied on real world ontologies, and (2) evaluate if the heuristics help less experienced ontology users to resolve the unsatisfiability of concepts.

8.2.1 Selecting Clusters of Heuristics

Benchmarking with real-life ontologies is obviously a convincing way to evaluate the quality of our heuristics. We therefore obtained a number of ontologies from the Internet, then tested the applicability of the heuristics. The problematic parts of axioms causing concepts to be unsatisfiable were identified by the fine-grained technique (see Chapter 4), and then were evaluated by the heuristics.

Before carrying out the usability evaluation with ontology users, we did a pre-selection of the heuristics in order to identify those heuristics which are useful for ontologies with common

OWL modelling errors, and heuristics which are useful for unsatisfiable ontologies merged from satisfiable ontologies. The two sets of heuristics are composed of:

1. "Common errors heuristics" are those capturing common errors, including those relating to sibling patterns and lengthy hierarchies. The disjointness and linguistic heuristics are also included.

2. "Merging heuristics" contain heuristics relating to depth of concepts in the hierarchy, as well as the disjointness and linguistic heuristics.

The disjointness heuristic appears in both sets because the subjects in the empirical studies were of the opinion that the 'disjointness' heuristic captures a good practice in the modelling of OWL ontologies, which is usually applicable to any ontology. The 'linguistic' heuristic is included in both sets because the subjects in our empirical studies stated it is generally applicable to unsatisfiable ontologies. Including these heuristics in both sets also has the advantage of making the difference between the confidence values of axioms higher, hence giving the experimental subjects a clearer recommendation for what should be modified. For example, if only the disjointness heuristic is applied to a set of axioms, only the disjointness axioms will be evaluated with confidence values (let us assume only one disjointness axiom is assigned with a positive confidence value), the rest will have the default value (i.e., zero), then the user might find it difficult to select the appropriate axiom for removal/modification among those axioms with zero value. The need for combining multiple heuristics is discussed further in Section 8.2.4.1.

In Table 8.1, the axioms causing a number of concepts to be unsatisfiable are evaluated by the "common error heuristics", the axioms with least confidence value are recommended to be removed/modified. The common error heuristics are applied in turn to each of the axioms and the value returned for each axiom is a cumulative value calculated as described in Section 6.3. The algorithm then recommends the axioms which have the lowest numeric value; several axioms will be recommended if they have the same value. So for concept ChemicalElement in Table 8.1 the following five axioms cause it to be unsatisfiable:

α_1: PublishedWork $\sqsubseteq \neg$ChemicalElement
α_2: VR_RelatedPublishedWork $\sqsubseteq$ NerveAgentRelatedPublishedWork
α_3: NerveAgentRelatedPublishedWork $\sqsubseteq$ PublishedWork
α_4: VR_RelatedPublishedWork$\doteq$ $\forall$ refersToPrecursor.VR_Precursor
α_5: domain (refersToPrecursor) = PublishedWork
The confidence value for those axioms is calculated as follows:
$conf_{disjSib}(\alpha_1) = \frac{6}{3(3-1)/2}$ =1;
$conf_{ling}(\alpha_2) = Sim(\text{VR\_RelatedPublishedWork}, \text{NerveAgentRelatedPublishedWork}) = \frac{30-10}{30} = 2/3$;
$conf_{ling}(\alpha_3) = Sim(\text{NerveAgentRelatedPublishedWork}, \text{PublishedWork}) = \frac{30-17}{30} = 0.43$;
$conf_{sib}(\alpha_4) = \frac{-4}{4+1} = -4/5$.
The confidence value of axiom α_5 is zero, as currently no heuristic is applicable to domain axioms. The confidence vector produced is

$$\langle 1, 0.67, 0.43, -0.8, 0 \rangle$$

and so the axiom α_4 is recommended as the one which should be removed/modified.

In Tables 8.1 and 8.2 we aim to evaluate if the set of heuristics is applicable for the ontologies; the problematic axioms causing the concepts to be unsatisfiable are not shown, for simplicity. We compare the suggestions given by the current version of RepairTab, which simply recommends the axiom with least confidence value, with the selection made by an ontology expert. In the 're-sult' column of the table, a $\checkmark$ is used to indicate that RepairTab's suggestion matches the expert's selection; a $\times$ is used to indicate that RepairTab gives a wrong suggestion to the user; `Pc` is used to indicate a partially correct selection, that means, RepairTab identifies a set of possible modifications which contains no incorrect selections, but does not identify exactly which modification should be made; `Pi` indicates a partially incorrect answer, that means, RepairTab identifies a set of modifications which contain both correct and incorrect modifications, so following RepairTab's advice the user could make an incorrect selection.

Table 8.1: Common error heuristics applied to ontologies

Ontology	Unsatisfiable concept	# of Axioms	Confidence vector on the set of axioms	Incorrect axioms(s) (suggested by RepairTab)	Incorrect axioms(s) (suggested by expert's view)	Result
CHEM.owl[a]	ChemicalElement	5	$\langle$ 1, 0.67, 0.43, -0.8, 0 $\rangle$	4th axiom	4th axiom	✓
	Person	5	$\langle$ 1, 0, 0.67, 0.43, -0.8 $\rangle$	5th axiom	5th axiom	✓
	Agent	6	$\langle$ 0, 0.67, 0.43, 1, -0.8, 0 $\rangle$	5th axiom	5th axiom	✓
	NerveAgentPrecursor	7	$\langle$ 0, 0.5, 0.43, 0.43, 1, -0.8, 0 $\rangle$	6th axiom	6th axiom	✓
	BlisterAgentPrecursor	7	$\langle$ 0.4, 0, 0.43, 0.43, 1, -0.8, 0$\rangle$	6th axiom	6th axiom	✓
Sweet-JPL.owl[b]	OceanCrustLayer	9	$\langle$ 0.25, 1, 0, 0, -0.25, 1, 0, 0, 0 $\rangle$	5th axiom	5th axiom	✓
Tambis.owl[c]	protein-part	7	$\langle$ 0, 0, 0, 0, 1, 1, 0 $\rangle$	1st, 2nd, 3rd, 4th or 7th axiom	4th and 7th axiom	`Pi`
	rna-part	7	$\langle$ 0, 0, 1, 0, 0, 1, 0 $\rangle$	1st, 2nd, 4th, 5th or 7th axiom	2nd and 7th axiom	`Pi`
	alkali-metal	4	$\langle$ 0, 0, 0, 1 $\rangle$	1st, 2nd or 3rd axiom	1st, 2nd and 3rd axiom	`Pc`
	metal	4	$\langle$ 0, 0, 1, 0 $\rangle$	1st, 2nd or 4th axiom	1st, 2nd and 4th axiom	`Pc`
	carbon	4	$\langle$ 0, 0, 0, 1 $\rangle$	1st, 2nd or 3rd axiom	1st, 2nd and 3rd axiom	`Pc`
University.owl[d]	CS_Department	5	$\langle$ 0, 0, 0, 1, -0.5 $\rangle$	5th axiom	5th axiom	✓
	Person	4	$\langle$ 0.625, 0, 0, 1 $\rangle$	2nd or 3rd axiom	2nd axiom	`Pi`
	AIStudent	4	$\langle$ 1, 0, 0, 0 $\rangle$	2nd, 3rd or 4th axiom	4th axiom	`Pi`
	Lecturer	3	$\langle$ 0, 1, 0 $\rangle$	1st or 3rd axiom	both 1st and 3rd axiom	`Pc`
	CS_Course	6	$\langle$ 0, 0, 0, 0, -0.5, 1 $\rangle$	5th axiom	5th axiom	✓
	CS_StudentTakingCourses	9	$\langle$ 1, 0, 0, 0, 0, 0, -0.5, 0.43 $\rangle$	8th axiom	8th axiom	✓
Koala.owl[e]	Quokka	4	$\langle$ 1, 0, 0, 1 $\rangle$	2nd or 3rd axiom	3rd axiom	`Pi`
	Koala	4	$\langle$ 0, 0, 1, 1 $\rangle$	1st or 2nd axiom	2nd axiom	`Pi`

[a]http://www.mindswap.org/ontologies/debugging/CHEM-A.owl

[b]http://www.mindswap.org/ontologies/debugging/buggy-sweet-jpl.owl

[c]http://protege.stanford.edu/plugins/owl/owl-library/tambis-full.owl

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

[e]http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

We now analyse Table 8.1 as follows:

1. RepairTab gives correct suggestions in the cases of CHEM.owl and Sweet-JPL.owl ontologies. It is suggested that this is because CHEM.owl contains common OWL modelling errors; the unsatisfiable concept in Sweet-JPL.owl was introduced by the modeller carelessly.

2. RepairTab gives partial answers for all unsatisfiable concepts in the case of the Tambis.owl ontology. The Tambis.owl ontology contains 144 unsatisfiable concepts (out of 395) due to an error in the transformation script used in the conversion process [91]. The script transforms the definition of concepts using equivalence axioms, and in fact, subsumption axioms should be used. Therefore, this set of heuristics fails to indicate the erroneous axioms precisely.

3. RepairTab gives partial answers for some unsatisfiable concepts in the cases of University.owl and Koala.owl ontologies. This is because the expressivity of these two ontologies is $\mathcal{SIOF}(\mathbf{D})$ and $\mathcal{ALCHON}(\mathbf{D})$ respectively; many unsatisfiable concepts are due to inverse properties, transitive properties, etc. However, our heuristics currently do not consider the characteristics of these properties, and cannot, therefore, be expected to identify errors due to them.

The result shows that the ‘common error heuristics’ give correct suggestions for ontologies with common OWL modelling errors. Currently, we aim to provide support for newcomers to OWL and Description Logics to debug ontologies. Note that common OWL modelling errors are usually made by the newcomers, and the heuristics are able to give them correct suggestions. In some cases, the heuristics are limited with respect to the expressivity of ontologies, and ontologies migrated from other formalisms, but no complete incorrect suggestion is generated.

Table 8.2: Merging heuristics applied to unsatisfiable ontologies resulting from merging two satisfiable ontologies

Ontology	Unsatisfiable concept	# of Axioms	Confidence vector on the set of axioms	Incorrect axioms(s) (suggested by RepairTab)	Incorrect axioms(s) (suggested by expert's view)	Result
SUMO.owl+CYC.owl	PhysicalQuantity	11	⟨ 0.62, 0.2, 0.25, 0.33, 0.5, -0.3, 0.12, 0.14, 0.16, 0.2, 0.25, 0.33 ⟩	8th axiom	8th axiom	✓
	Relation	8	⟨ 0.2, 0.25, 0.33, 0.2, 0.25, 0.33, 0.5, -0.3 ⟩	8th axiom	8th axiom	✓
	TruthValue	10	⟨ 0.2, 0.25, 0.33, 0.2, 0.16, 0.2,0.25,0.33,0.5,-0.3 ⟩	10th axiom	10th axiom	✓
CONFTOOL+EKAW[a]	Camera_Ready_Event	6	⟨ 0.56, 0.2, 1, 0, 0, 0.7 ⟩	4th or 5th axiom	6th axiom	×
	Poster_Paper	3	⟨ 1, 0.54, 0.45 ⟩	3rd axiom	3rd axiom	✓
EKAW+PCS[b]	ConferenceMember	5	⟨ 0, 1, 0.25, 0, 0.625 ⟩	1st or 4th axiom	5th axiom	×
	Author	4	⟨ 0.73, 0, 1, 0.12 ⟩	2nd axiom	1st axiom	×
EKAW+CRS[c]	Conference_Session	6	⟨ 0, 0.32, 1, 0.2, 0.52, 0 ⟩	1st or 5th axiom	5th axiom	Pi
	Workshop	3	⟨ 0, 1, 0 ⟩	1st or 3rd axiom	3rd axiom	Pi
	Participants	4	⟨ 0, 0.32, 0.1, 0.61 ⟩	1st axiom	4th axiom	×

[a]Two ontologies are merged by Falcon-AO. Available at http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html

[b]Two ontologies are merged by COMA++. Available at http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html

[c]Two ontologies are merged by HMatch. Available at http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html

In the second experiment, we studied a number of unsatisfiable ontologies which had resulted from merging two satisfiable ontologies. From Table 8.2, we notice that RepairTab gives correct suggestions in the case of SUMO.owl+CYC.owl (which was merged from SUMO.owl and part of CYC.owl), but gives some incorrect suggestions with the other merged ontologies. This is because the systems (namely COMA++ [108], Falcon-AO [82] and HMatch [26]) create the union of two ontologies by adding additional (and sometimes incorrect) axioms. For example, given two concept names, ConferencePaper and ConferenceMember, COMA++ generates a subsumption relation between them due to their lexical similarity. Our linguistic heuristic also fails to detect this error because we also use lexical similarity, which is too primitive to verify the subsumption relationships.

The linguistic heuristic results in wrong suggestions, because the unsatisfiable ontologies are created by joining pairs of ontologies using automatically generated mappings. Earlier results show that, the 'merging heuristics' are able to give correct suggestions if ontologies are simply merged without additional axioms.

8.2.2 Usability Study

For the purpose of evaluation, we decided to compare our heuristics with existing approaches. SWOOP[4] is a stand-alone ontology editor offering ranking strategies for evaluating the importance of problematic axioms, namely *arity* of axioms, *impact* of removal and *usage* of ontology elements (we have described their strategies in detail in Section 3.4.3). It is useful to compare our plugin with SWOOP. However, we consider that SWOOP's graphical user interface is less friendly than Protégé's; additionally ontology users may find it difficult to edit and understand SWOOP's description logic notations. Most importantly, if we had some users using SWOOP, differences in the perceived usefulness might result from the different environment rather than the different heuristics. Due to these user interface issues it is easier for us to compare the performance of our plugin with another plugin to the same editor, and it is for this reason that we have embedded SWOOP's strategies into our plugin using the implementers' original default weights for the strategies (where the weight of arity, impact and usage are 0.9, 0.7 and 0.1, respectively) [91]. As we are unclear about the relative benefits of the "common errors" or "ontology merging" heuristics, we have assumed that they have equal weighting; i.e., weights used to create the recommendations presented to the user when these heuristic sets are used. Our hypotheses for the usability study were:

1. The confidence information is helpful for subjects to resolve unsatisfiable concepts. That means, the subjects will take less time to understand and fix the errors, when compared to subjects without heuristic support.

2. The "common errors heuristics" are helpful when debugging unsatisfiable ontologies which contain common mistakes. That means, the subjects using these heuristics would achieve better results in less time compared to other subjects using other heuristics or no heuristics.

3. The "merging heuristics" are helpful when debugging ontologies which are merged from

[4]http://www.mindswap.org/2004/SWOOP

two satisfiable ontologies. That means, the subjects using these heuristics would achieve better results in less time compared to other subjects using other heuristics or no heuristics.

We conducted a usability study with two types of unsatisfiable ontologies, namely ontologies containing common OWL modelling mistakes, and unsatisfiable ontologies merged from satisfiable ontologies. The first type of ontologies (i.e., those with common mistakes) ideally would fulfil the following conditions:

1. they should be interestingly axiomatised, i.e., containing axioms like disjointness, role restrictions, concept definitions, and so on, and should not be simply taxonomies;

2. the domain of the ontologies should be easily understood by subjects;

3. they contain unsatisfiable concepts which could be difficult for non-expert users to debug.

The CHEM.owl and Mad_Cow.owl[5] ontologies were chosen for evaluation. For the CHEM.owl ontology, Table 8.1 shows that the heuristic set gives correct suggestions; it contained 37 unsatisfiable concepts, most of them were due to their unsatisfiable superconcepts. Mad_Cow.owl ontology was modified to contain five unsatisfiable concepts, these unsatisfiabilities were realistic based on commonly observed errors as enumerated by Rector et al. [125].

The second type of ontologies (resulting from ontology merging) need to fulfil the following conditions:

1. the individual pre-merged ontology should be semantically correct;

2. the concepts in taxonomies should contain multiple parents; the hierarchies should be large; disjointness should exist between some concepts;

For the second type of ontologies, SUMO.owl and part of CYC.owl were merged and chosen for evaluation, instead of other merged ontologies, such as CONFTOOL+EKAW, generated by Falcon-AO [82], COMA++ [108], or HMatch [26]. This is because SUMO.owl and CYC.owl, which are upper ontologies, are topic-related and CYC.owl contains disjointness axioms; CONFTOOL+EKAW, EKAW+PCS, and EKAW+CRS ontologies contain inconsistencies which could be too trivial for the subjects, e.g., $\mathsf{ConferencePaper} \sqsubseteq \mathsf{ConferenceMember}$ is easy to spot. In our study, we simply removed the unique name-spaces from both SUMO.owl and CYC.owl and syntactically merged the two ontologies (as in [130]) to create the SUMO.owl+CYC.owl ontology. As both ontologies contain a very large number of concepts, the merged ontology contains numerous unsatisfiable concepts, and so we simplified the resulting merged ontology by removing parts of it. As a result, only six unsatisfiable concepts were left; the subjects in the study were then not affected by the loading time of the ontology and the performance of the system.

Twenty subjects, who were undergraduate students in the Computing Science Department at the University of Aberdeen, were chosen for the evaluation. They had basic knowledge of OWL ontologies and Description Logics; they were also familiar with Protégé. They were divided into four groups to debug the same set of ontologies using different sets of heuristics in the plugin (and in one case no heuristics).

[5]http://www.cs.man.ac.uk/∼horrocks/OWL/Ontologies/mad_cows.owl

- **Group A**: Subjects in this group were provided with recommendations based on the heuristics proposed by the SWOOP system.
- **Group B**: In this group, the plugin evaluated the confidence of axioms by using the "common error heuristics" which include disjointness, sibling pattern, length of hierarchy and linguistic.
- **Group C**: In this group, the plugin evaluated the confidence of axioms by using the "merging heuristics": disjointness, depth of concepts in the hierarchy and linguistic heuristics.
- **Group D**: Subjects in this group received no recommendations about what axioms were likely to be faulty.

The usability study was conducted as follows:

1. A sample ontology was used to demonstrate the features of the plugin.
2. Each subject was given the same set of ontologies. None of the subjects had seen these ontologies before. The subject was asked to resolve the errors with the assigned plugin which contained one of four different sets of heuristics.
3. Before the task, subjects were reminded that they should not merely remove axioms to try to get rid of the inconsistencies; instead, they should try to rewrite the axioms in a way that preserves as much as possible of the original ontology.
4. After debugging each ontology, the subjects were asked to explain what they understood about the cause of the unsatisfiable concepts, and how they selected the axioms to modify.[6] They were also asked to rate the usefulness of the plugin for each ontology.
5. Additionally, at the end of the study, we also asked the subjects for their comments on using the plugin, and how the plugin could be improved. The detail of the questionnaires is given in Appendix D.
6. The time taken by subjects for resolving each ontology was recorded by the plugin. The changed ontologies were also recorded so that their quality could be assessed by the investigator.

8.2.3 Results

In the following, we firstly compare the observations made by the groups with and without heuristics, and then analyse the performance of groups with different heuristics.

Groups With and Without Heuristics

We now lump together in one set all those subjects who were provided with confidence values (i.e., Groups A, B and C); based on the results of the experiment we could identify three different

[6]From this we are able to infer whether the subjects solved the tasks themselves or relied on the advice of the plugin. This information was deduced on a task-by-task basis by the investigator.

categories of subjects within this set. These three categories emerged because the students in the class actually have very different levels of ability.

1. If subjects understood the reasons for the unsatisfiable concepts and the domain of the ontologies, then two different outcomes were possible:
   - For some of these subjects the heuristics were not helpful, as they could apply their background knowledge to judge which axioms should be modified.
   - For some of these subjects the heuristics were helpful as they guided them towards the axioms which needed to be modified. This helped them achieve correct modifications in shorter time.
2. If subjects understood the domain of the ontologies, but not the reasons for the unsatisfiable concepts completely, then the heuristics could increase their confidence in the modification which they were making:
   - If their selection of axioms for modification conformed to the system's recommendation (i.e., the lowest confidence value was selected), then the subjects would modify the recommended ones.
   - When the subjects were wondering which of two axioms to modify, they could use the system's recommendations.
3. If subjects understood neither the reasons for the unsatisfiable concepts nor the domain of the ontologies, then two different outcomes were possible:
   - For some of these subjects the heuristics were helpful, as the heuristics could guide them towards the axiom which would be the best to modify.
   - For some of these subjects the heuristics were not helpful, as they were not sure if they could rely on the tool's recommendation; they didn't make any changes to the ontology as they felt unsure about what to do.

In comparison, most subjects in Group D said that the ontologies were difficult to debug, especially Mad_Cow.owl and CHEM.owl ontologies. We believe these ontologies, which contain common OWL modelling errors, are very difficult for the subjects. The subjects received no guidance on which axiom(s) are erroneous; they took longer to analyse the meaning of every single axiom, and fewer correct changes were made, compared to other groups.

Groups with Different Heuristics

We now describe the performance in Groups A, B and C to analyse how useful the assigned heuristics were for them to resolve the errors.

Observations made about Group A (SWOOP control group):

1. For the Mad_Cow.owl and CHEM.owl ontologies, some subjects applied their background knowledge to solve the problems without referring to the confidence values; some subjects

applied their background knowledge in analysing the problems, and confirmed their decision with the plugin.

2. For the merged ontology, some subjects relied on the confidence values, they made changes without analysing the axioms. They did not remove the axioms whose removal would cause a big impact on the ontology.

Observations made about Group B (common mistakes):

1. Similarly to Group A, for the Mad_Cow.owl and CHEM.owl ontologies, Group B performed better than Group C and D. Some subjects took the confidence values as confirmation; some did not rely on the given values to select which axioms to modify.

2. For the SUMO.owl+CYC.owl ontology, the resulting confidence values for all the axioms were quite close, except for the disjoint axioms, showing that all axioms were almost equally wrong. Most subjects applied their background knowledge to justify their modifications.

Observations made about Group C (merging heuristics):

1. The confidence values were not useful for the ontologies with common mistakes, because the confidence values were very close. The subjects explained that they could not rely on the values to select axioms for modification. Though the merging heuristics did occasionally produce large differences in confidence values, this tended to be uninformative to the subjects as it was often very obvious to the subjects based on their background knowledge which axiom should be removed. On the other hand, they usually took more time to complete the tasks for Mad_Cow.owl and CHEM.owl compared to Groups A and B, and the quality of resulting ontologies was slightly worse than Group A and B (as assessed manually).

2. For the merged ontology, the recommendations were quite helpful, as axioms were assigned with distinctive values, especially, when the subjects were not familiar with the domain. All the subjects made correct changes in less time compared with the other groups.

Table 8.3 shows the results for the four groups working with the three ontologies. We took the average of the times for each group. We reviewed each of the hypotheses and the results of the experiment and have come to the following conclusions: (1) If we compare Groups A, B and C with Group D, we can see that the subjects in Groups A, B and C took less time to make the correct changes. We suggest that this is because the confidence values guided them towards axioms to modify. This result is in agreement with the first hypothesis (see Section 8.2.2). (2) The subjects in Group C performed better on SUMO.owl+CYC.owl when compared with the other groups (i.e., they achieved better results in less time). Some subjects in Group A were mislead by the confidence values and made incorrect changes; some in Group B could not rely on the confidence values which were quite close. This result is in agreement with the third hypothesis. (3) When resolving ontologies with common mistakes, Group A and B noticeably outperformed Groups C and D for similar reasons as in (1). Therefore we were unable to find evidence for the second hypothesis, although the weakened form of it is supported (Group B is better than Group C and D). As shown in Table 8.3, the subjects in Groups A and B had similar performance. We need extensive study to evaluate if Group B can really achieve an improvement over Group A, but we are fairly clear that the approach is at least comparable.

Ontology / Group	A	B	C	D
Mad_Cow.owl				
No. of subjects who understood the errors	4/5	4/5	3/5	3/5
No. of subjects who made correct answers	3/5	4/5	3/5	2/5
Average Rating of Usefulness	3.8	4	3	-
Average Time Taken	15.2 min	14.8 min	23.4 min	27.8 min
CHEM.owl				
No. of subjects who understood the errors	3/5	4/5	2/5	2/5
No. of subjects who made correct answers	3/5	3/5	2/5	2/5
Average Rating of Usefulness	3.8	3.8	2.6	-
Average Time Taken	19 min	17.8 min	25.6 min	29.5 min
SUMO.owl+CYC.owl				
No. of subjects who understood the errors	5/5	5/5	5/5	5/5
No. of subjects who made correct answers	3/5	3/5	5/5	3/5
Average Rating of Usefulness	3.8	3.4	4.4	-
Average Time Taken	14.6 min	15.3 min	9.4 min	21 min

Table 8.3: Experimental results of resolving unsatisfiable ontologies (A = SWOOP, B = Common Mistakes, C = Merging Heuristics, D = No Heuristic)

8.2.4 Discussion

In this section, we address the scope and limitations of our approach. We firstly discuss the methods and results of the evaluation of the heuristics, and present improvements on our evaluation and proposed approach.

8.2.4.1 Evaluation of Heuristics

The above study grouped heuristics into three sets for evaluation (i.e., SWOOP, common errors, merging). Other alternatives could have been chosen; we now explain why the following two options were not studied: (1) evaluate each heuristic individually. If only one heuristic is applied to ontologies, such as disjointness, then the confidence values for some axioms may be positive, but many axioms will have zero confidence (i.e., the non-disjointness axioms). In this case the user has no clear recommendation for which axiom to modify. Similarly, if we only apply the depth of concepts heuristic to a hierarchy, then the confidence values of axioms will be very close, that means, the axioms are almost equally erroneous. The results of a single heuristic will not be useful for subjects (as shown above: we saw that common error heuristics were not helpful for the SUMO.owl+CYC.owl ontology). However when heuristics are combined, we often obtain different confidence values assigned to each axiom, hence giving a clearer recommendation for which one should be modified. (2) different combinations of heuristics, or different weights on heuristics. For example, given n heuristics, we can create 2^n combinations of heuristics, at the same time, each individual heuristic could be assigned with different weights. In such cases we would need a very large number of unsatisfiable ontologies for a complete study. It might not be practical to carry out such a huge study. Instead, when there are more unsatisfiable ontologies available on the Web in the future, we are interested in evaluating the interaction between heuristics.

8.2.4.2 Generating Unsatisfiability

Regarding the issue of the lack of unsatisfiable ontologies currently in the public domain (for evaluation purposes), it could be argued that it is possible to make random changes (such as adding disjointness, multiple parents, quantified restrictions) to satisfiable ontologies so that unsatisfiable concepts are generated. However, the mistakes generated by the system might be too obvious/easy in which case the subjects would not need the support of our heuristics, to resolve the problems. We now try to generate unsatisfiable concepts by adding domain, multiple parents or disjointness with the following example.

Example 40 *Let us take the Pizza.owl*[7] *from the Protégé OWL tutorial as an example.* $\mathsf{IceCream}$ *is defined as:*
1. $\mathsf{IceCream} \sqsubseteq \exists\, \mathsf{hasTopping}.\mathsf{FruitTopping}$,
2. $\mathsf{IceCream} \sqsubseteq \neg\, \mathsf{Pizza}$
A system adds a domain axiom to result in an unsatisfiability:
3. $\mathit{domain}(\mathsf{hasTopping}) = \mathsf{Pizza}$

Now $\mathsf{IceCream}$ becomes unsatisfiable. However, as our subjects have basic knowledge of OWL-DL, it could be easy for them to spot that if $\mathsf{IceCream}$ has the property $\mathsf{hasTopping}$, the domain of $\mathsf{hasTopping}$ should not be Pizza. If the problematic axioms are easy to spot, then the support for suggesting appropriate axioms for removal/modify is no longer useful.

Example 41 *For another example,* $\mathsf{IceCream}$ *is unsatisfiable due to the following axioms:*
1. $\mathsf{Pizza} \sqsubseteq \neg\, \mathsf{IceCream}$,
2. $\mathsf{IceCream} \sqsubseteq \exists\, \mathsf{hasTopping}.\mathsf{FruitTopping}$,
3. $\mathit{inverseFunctional}(\mathsf{hasTopping})$,
4. $\mathsf{hasTopping}$ *inverse* $\mathsf{isToppingOf}$,
5. $\mathit{range}(\mathsf{isToppingOf}) = \mathsf{Pizza}$

Because this example is generated by humans, it could be more challenging for our subjects. However, we need more sophisticated algorithms to generate such challenging errors automatically. In future work, we can borrow techniques to generating concepts unsatisfiability in OWL ontologies from the methods for testing propositional satisfiability (SAT) decision procedures [53; 48]. Two random generation procedures were used in the SAT tests, which are devised by Giunchiglia and Sebastiani [53] and Hustadt and Schmidt [85]. Both procedures use a number of parameters to control the size and complexity of the generated expressions; the Hustadt and Schmidt procedure is designed to eliminate trivially unsatisfiable formulae and so generates consistently more difficult problems.

8.2.4.3 Correctness of Different Types of Axioms

As we mentioned in Chapter 5, some subjects' heuristics are contradictory. For example, some subjects stated that ontology experts are good at modelling ontologies; they usually do not introduce errors and understand the whole ontology, whereas non-sophisticated users usually have

[7] http://www.co-ode.org/ontologies/pizza/2006/07/18/pizza.owl

misunderstanding and need help from tools/systems to resolve errors. Therefore, knowledge of the common mistakes in modelling OWL should be applied to resolve inconsistencies.

Referring to the example in Figure 6.1 in Section 6.3.6, the subjects explained that the axiom $\mathsf{C3} \sqsubseteq \forall\ \mathsf{has\text{-}R}.\neg\mathsf{D2}$ is likely to be erroneous, as it is modelled differently from the other siblings, and ontology users usually misuse the universal restrictions rather than existential restrictions as the default quantifier.

In contrast, some subjects in the empirical studies stated that disjointness or universal restrictions axioms are unlikely to be asserted by mistake. Therefore, in our sibling example, the axiom $\mathsf{C3} \sqsubseteq \forall\ \mathsf{has\text{-}R}.\neg\mathsf{D2}$ is likely to be correct. From our subjects' experience, ontology modellers usually assert disjointness or universal restrictions after careful consideration, hence they are unlikely to be incorrect; in contrast, modellers often make mistakes with the subclass-of relation. The following example is used to explain their claim:

Example 42 *For the example used in [57],* Agent *subsumes two classes,* Animal *and* Social_Entity*. As any instances of* Agent *might cease to be an agent in the future, the property is then anti-rigid ($\sim R$). In contrast, the property of being an* Animal *or* Social_Entity *is rigid ($+R$). A property with $\sim R$ cannot subsume a property with $+R$. Any* Agent *could cease to be an agent, and since all animals are agents, this would mean the changed entity would cease to be an animal (not allowed). Therefore, the subsumption relations are incorrect. The modeller intends to mean, not that all animals are agents, but that animals* can be *agents. This is a very common misuse of subsumption, often employed by object-oriented programmers.*

The above examples show that an arbitrary type of axiom can be one of the common mistakes by a novice, or can be unlikely to be stated by mistakes by an expert. With these diverse opinions on the correctness of axioms, it seems that we could assign different precedence levels to different types of axioms based on the level of expertise of the user. If an ontology editor has access to provenance information, e.g., the level of expertise of the user, or information about the process by which the ontology is constructed. If the ontology built by a novice is unsatisfiable, then the "common errors" heuristics should be used. On the other hand, if the unsatisfiable ontology is formed by merging two satisfiable ontologies, then the "merging heuristics" should be used. This is further discussed in Section 9.4.1.

8.2.4.4 Improvement of the Functionalities

From the observations during our studies, we noticed that some subjects were not sure if they could rely on the tool's recommendation. To improve this, the recommendation for selecting which axioms to modify should be explained in a way that is understandable to the users. For a disjoint axiom with a high confidence value, the tool can explain to the user that it is good practice that sibling concepts are usually disjoint; for a disjoint axiom with a low confidence value, the tool can explain to the user that it is unusual for two concepts which are far from their common ancestor to be disjoint. If certain heuristics are not suitable for his ontologies, he can adjust the weights of the heuristics accordingly.

8.2.5 Conclusion

We incorporated the heuristics obtained from the empirical studies into an ontology debugging tool. We can conclude four issues from the usability study results: (1) with the heuristic support, the users were guided towards the axioms which would be the best to modify, achieve correct modification in shorter time; (2) the heuristics could increase the users' confidence in the modification which they are making; (3) the common error heuristics were useful for dealing with ontologies with common mistakes; (4) the merging heuristics are useful for dealing with integrating ontologies. These results are consistent with the assumption we made in Chapter 5, namely that the heuristics which have been obtained in the studies from subjects working with abstract stimuli are in fact applicable when subjects work with concrete stimuli. Our result is an important starting point in ontology debugging; future work will focus on the limitations addressed in the above.

8.3 Evaluation of the Fine-Grained Approach

In this section we describe a usability evaluation which addresses the benefits of the fine-grained approach described in Chapters 4 and 7; the results are compared with existing debugging tools. Next, we present the performance evaluation of our prototype using a set of satisfiability tests and compare it with an existing DL reasoner.

8.3.1 Usability Evaluation

To illustrate the practical use of our proposed approach, we have extended our Protégé plugin, 'RepairTab', for supporting ontology repair. For the purpose of evaluation, we decided to compare RepairTab with other ontology debuggers. There were two obvious choices, namely, 'OWLDebugger'[8] and 'SWOOP'[9]. OWLDebugger is another Protégé plugin, which provides explanations for unsatisfiable concepts. SWOOP is a stand-alone ontology editor. We are interested in two main functionalities in SWOOP [87]. The first one is the explanation of unsatisfiability – it pinpoints the problematic axioms for unsatisfiable concepts, and is able to strike out irrelevant parts of axioms that do not contribute to the unsatisfiability. The second one is the ontology repair service – SWOOP displays the impact on ontologies due to the removal of axioms. When an axiom is removed, it shows the fixed and remaining unsatisfiable concepts, as well as the lost and retained entailments.

We conducted a usability study with three ontologies and three groups of subjects. Fifteen subjects, who were under taking a taught Master's degree in the Computing Science Department at the University of Aberdeen, were chosen for the evaluation. They had knowledge of OWL ontologies and Description Logics; they had experience of using both Protégé and SWOOP. None of the subjects had seen these ontologies before, and they were divided into three groups to debug the same set of ontologies using one of the three tools. Ideally, ontologies in our evaluation would fulfil the following conditions:

[8]http://www.co-ode.org/downloads/owldebugger/
[9]http://www.mindswap.org/2004/SWOOP/

1. The expressivity of ontologies is in $\mathcal{ALC}$;
2. They should be interestingly axiomatised, i.e., containing axioms like disjointness, existential, definitions, etc., and not just taxonomies;
3. The domain of the ontologies can be easily understood by subjects;
4. They are available on the Web;
5. They contain unsatisfiable concepts which could be difficult for non-expert users to debug.

The Mad_Cow[10], Bad-food.owl[11] and University.owl[12] ontologies are available on the Web and were chosen for evaluation. Both the Mad_Cow.owl and Bad-food.owl ontologies each contain one unsatisfiable concept. University.owl was simplified into $\mathcal{ALC}$ format. The simplified version contains 12 unsatisfiable concepts which were sorted based on the number and size of the MUPSs of the concepts. Our hypotheses for this usability study were:

1. The explanation function of RepairTab, which highlights parts of the axioms causing the unsatisfiability, helps users to resolve the unsatisfiability. This is a relative advantage of RepairTab when compared with OWLDebugger and SWOOP.
2. The subjects using RepairTab will take less time to understand the source of unsatisfiabilities and resolve them, compared with OWLDebugger and SWOOP.
3. RepairTab's list of lost entailments helps subjects decide which change(s) should be made in order to minimise the impact on the ontologies.
4. RepairTab's list of helpful changes provides useful (as rated by subjects) suggestions for subjects to add axioms back to the ontologies in order to minimise the impact of the changes on the ontologies.
5. RepairTab's list of harmful changes provides useful (as rated by subjects) guidance for subjects about which changes should not be made in order to prevent more unsatisfiable concepts being created.

The usability study was conducted as follows: Three groups of subjects were each given a tutorial on OWLDebugger, RepairTab and SWOOP, demonstrating their key features. A detailed walk-through of the relevant explanation and debugging functions was given using a sample ontology.

1. **Group A** was assigned to resolve all unsatisfiable concepts in the three ontologies using RepairTab.
2. **Group B** was assigned to resolve the same set of unsatisfiable concepts using the OWLDebugger plugin with the FaCT++[13] reasoner.
3. **Group C** was assigned to resolve the same set of ontologies using SWOOP.

[10]http://www.cs.man.ac.uk/∼horrocks/OWL/Ontologies/mad_cows.owl
[11]http://www.mindswap.org/dav/ontologies/commonsense/food/foodswap.owl
[12]http://www.mindswap.org/ontologies/debugging/university.owl
[13]http://owl.man.ac.uk/factplusplus/

The subjects in the three groups were asked to answer a survey. For each ontology, they were asked if they understood the cause of the unsatisfiable concepts, which axioms were changed, and how many changes were made etc. After debugging each ontology, they were also asked to rate the usefulness of the facilities provided by tool used on a 5 point scale where 5 is 'very useful' and 1 corresponds to 'useless'. For Group A, the subjects were also asked how useful the explanation function, the lost entailments, helpful and harmful changes facilities were, and how many helpful changes they had selected to add back to each of the three ontologies. For Group C, the subjects were also asked about the usefulness of the explanation and repairing functionalities provided by SWOOP. At the end of the study, we also asked the subjects for their comments on the tool used, and how it could be improved. The time taken by each subject for resolving the unsatisfiability in each ontology was recorded. The modified ontologies were also recorded for analysis by the experimenter. The details of the survey are given in Appendix E.

Observations made about Group A who used RepairTab:

[**1**] The highlighted parts of axioms helped them to analyse the source of errors. For example, it is easy to see that the part of the axiom '$\mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg \mathsf{Animal})$' in the axoim $\mathsf{Vegetarian} \doteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg \mathsf{Animal}) \sqcap \forall \mathsf{eats}.(\neg \exists \mathsf{part\_of}.\mathsf{Animal})$ is not responsible for the problem of $\mathsf{mad\_cow}$ in Mad_Cow.owl. (see Figure 8.2)

[**2**] All subjects previewed the information of the impact of removal before removing the axioms. They usually considered the impact of all possible removals, analysed the lost entailments, helpful and harmful changes, and then decided on the changes to be applied. Most of them added the helpful changes back to the ontology. For example, some subjects who tried to remove $\mathsf{Vegetarian}$ from the axiom $\mathsf{Cow} \sqsubseteq \mathsf{Vegetarian}$, were surprised to see that $\mathsf{Cow} \sqsubseteq \mathsf{Animal}$ and $\mathsf{Cow} \sqsubseteq \forall \mathsf{eats}.(\neg \mathsf{Animal})$ could be added back to the ontology.

[**3**] Three subjects tried to remove disjoint axioms (e.g., $\mathsf{Person} \sqsubseteq \neg \mathsf{University}$ in University.owl), but they changed their mind when RepairTab pointed out that this would result in many lost entailments.

[**4**] For a large number of unsatisfiable concepts, the ordering of unsatisfiable concepts was found to be useful by the subjects. The concepts were sorted based on the number and size of the MUPS. Note that when a concept is unsatisfiable, all of its subconcepts are also unsatisfiable. Students were surprised that when the unsatisfiable concept Person in University.owl was resolved, most of its subconcepts were resolved as well.

Results and overall comments from the subjects in Group A.
For those subjects who understood the problems but had no idea what changes should be made, the impact of removal and suggested changes were rated to be very useful. On the other hand, for those subjects who already had an idea what changes should be made, the impact and suggestions of changes were not useful; for example, if a subject wants to make complex changes, such as changing role restrictions or creating new concepts, then our plugin does not support these changes. The list of harmful changes was rated as 3 on average. This is because the subjects who understood the causes of problems, already knew what changes should not be made.

The overall comments on our plugin were that it is useful for resolving inconsistencies, but that the presentation of problematic axioms could be more user friendly, such as using natural language. Two subjects thought the presentation of problematic axioms was too formal, and they took time to analyse the meaning of those axioms. For example, Protégé presents disjoint concepts in a `Disjoints` table, but a disjoint axiom is presented in our plugin as '$C \sqsubseteq \neg D$'.

Observations made about Group B who used OWLDebugger:

[**1**] The subjects were led through various debugging steps, and provided with suggestions for which concepts should be debugged. For example, in University.owl, subjects were pointed to debug Person first (as Person is the root unsatisfiable concept), when the concepts Faculty, or TeachingFaculty were chosen to be debugged.

[**2**] All subjects firstly analysed the natural language explanations, and then focused on the contradicted conditions which were highlighted. However, in some cases, the explanation of the unsatisfiable concepts was oversimplistic when the cause of the unsatisfiability was too complex to explain by the current algorithm. For example, the debugger explained that Mad_Cow was unsatisfiable because it was a subconcept of Cow.

[**3**] Most subjects usually did not remove the contradictory conditions directly to resolve the errors, instead they took time to analyse the reason for the unsatisfiability, and then modified the definitions of concepts, and ran the reasoner to check the satisfiability. For example, a necessary and sufficient condition was changed to a necessary condition, or a class had two new subclasses added to it, the definitions of the class were split and moved down to the new subclasses. However, one subject made some changes which caused more unsatisfiable concepts.

Results and overall comments from the subjects in Group B.
Most subjects thought the plugin was useful because it indicated which conditions contradict with each other; the clash information was also shown in quasi-natural language. Debugging steps were provided to suggest which concepts should be debugged. For example, in University.owl, subjects were pointed to debug CS_Student when the concept AIStudent was chosen to be debugged. However, sometimes the explanation of the unsatisfiable concept was oversimplistic when the cause of the unsatisfiability was too complex to explain. Most importantly, most subjects usually did not remove the contradictory conditions directly to resolve the errors, instead they tried to modify the definition of concepts and run the reasoner to check for the consistency.

Observations made about Group C who used SWOOP:

[**1**] In some cases, the function of 'striking out irrelevant parts of axioms' cannot really strike out the irrelevant parts of axioms, some subjects found the explanation not helpful. For example, in Mad_Cow.owl in Figure 8.2, it only strikes out 'brain' from the axiom of mad_cow, but fails to strike out the irrelevant part '$\mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg \mathsf{Animal})$' in the axiom $\mathsf{Vegetarian} \doteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.(\neg \mathsf{Animal}) \sqcap \forall \mathsf{eats}.(\neg \exists \mathsf{part\_of}.\mathsf{Animal})$ (the reason for this is discussed below).

[**2**] All subjects focused on debugging the root unsatisfiable concepts, and ignored the derived ones. The derived ones are usually resolved automatically when the root ones are resolved.

[**3**] When the subjects tried to remove an axiom, they usually previewed the information of the impact of removing axioms, and checked the fixed and remaining unsatisfiable concepts, together with the lost and the retained entailments. Some subjects removed axioms which would cause the least lost entailments.

Results and overall comments from the subjects in Group C.

(i) *Explanation Function*: One subject said the function of 'striking out irrelevant parts of axioms' was useful in general; it helps the subject focus on certain parts of axioms. Two subjects thought the function was not useful because it only works in a few cases; the subjects still had to analyse the reasons for unsatisfiabilities. Two subjects thought the function was confusing, because in some cases, it did not strike out all of the irrelevant parts (mad_cow in Mad_Cow.owl is an example), sometimes, it struck out the relevant parts of axioms. Person in University.owl is an example, in which the whole right-hand side of an axiom was struck out; this misled the subjects to think that the problematic axiom was not responsible for the unsatisfiability.

We have analysed the reason for the above result. The explanation functionality was implemented by Kalyanpur et al. [87] who published a mathematical description of the algorithm to capture precise justifications, which determines which parts of the asserted axioms are irrelevant for the unsatisfiability of concepts. However the implementation in the latest version of SWOOP 2.3 Beta 3 is incomplete with respect to the published algorithm.

(ii) *Repairing Function*: Most subjects thought the tool was useful because (1) it separates the root and derived unsatisfiable concepts, so that they can focus on only debugging the root ones; (2) it enables the user to remove different axioms and preview the impact of removal before committing the change; (3) it displays the lost and retained entailments when axioms are removed. Additionally, in some cases, the tool provides `(Why?)` hyperlinks which explain why entailments are lost and retained. However, there is no explanation for the fixed and remaining unsatisfiable concepts.

Two subjects thought that SWOOP's repair function is limited to removal of the whole axioms, and changing certain parts of the axioms is not supported. Removing whole axioms will unnecessarily cause additional information lost. For the Mad_Cow.owl example, one subject claimed the definition of mad_cow was modelled poorly. If the definition axiom of mad_cow is completely removed, then all information about mad_cow will be lost. This is not a desired change for the subjects, though the ontology becomes satisfiable.

8.3.1.1 Analysis of Results

Table 8.4 shows the results for the three tools used by the subjects. We took the average of the times for each group to complete the tasks. As some tools do not provide certain functionalities, those ratings are not included in the table. As can be seen from the Table 8.4, firstly, the explanation function (i.e., highlighting the problematic parts of axioms) of RepairTab was rated to be more useful than SWOOP and OWLDebugger in two examples, but less useful on the Bad-food.owl ontology compared with OWLDebugger. Secondly, the subjects in Group A took less time to resolve the unsatisfiability than Group B; Group A had similar performance with Group C. Therefore, we cannot verify the first and second hypothesis currently. Both the lost entailments and helpful changes were rated to be useful overall, the ratings were in agreement with the third and fourth hypotheses. However, the harmful changes are less useful relatively, therefore the final hypothesis was falsified.

	Mad_Cow			Bad-food			University		
Group	A	B	C	A	B	C	A	B	C
Average Time Taken (in mins)	5	8.8	6.9	6.4	6.8	6.0	10.2	16.3	11.5
No. of subjects who understood the errors	5/5	4/5	3/5	3/5	2/5	3/5	0/5	0/5	0/5
Rating of Explanation Function	5	4	2.6	3.5	4.5	3.5	5	4	2.6
Rating of Lost Entailments (RepairTab)	5	-	-	4	-	-	5	-	-
Rating of Lost & Retained Entailments (SWOOP)	-	-	3	-	-	4	-	-	5
Rating of Fixed & Remaining Unsat. Concepts (SWOOP)	-	-	3	-	-	3	-	-	5
Rating of Helpful Changes (RepairTab)	5	-	-	4	-	-	4	-	-
Rating of Harmful Changes (RepairTab)	4	-	-	2.5	-	-	2.5	-	-

Table 8.4: Results of Debugging Ontologies (A = RepairTab, B = OWLDebugger, C = SWOOP)

We now analyse the subjects' performance for each ontology.

For Mad_Cow.owl, more subjects using RepairTab understood the error than those using OWLDebugger or SWOOP. It is suggested that this is because the problematic axioms of mad_cow were highlighted by RepairTab, and so the subjects understood the error quickly. However, it is difficult to resolve the problem correctly. The subjects in Group A usually resolved the error by removing part of an axiom and then adding the helpful changes suggested by the plugin; the subjects in Group B had to explore changes to the definitions of concepts or add extra subconcepts for cow (e.g., to have Normal_Cow as a sibling of mad_cow), one subject also triggered additional unsatisfiable concepts. Two subjects in Group C failed to understand the cause of the unsatisfiable mad_cow, because some irrelevant parts of axioms were not struck out, this led the subjects to think that the irrelevant parts were responsible for the unsatisfiability.

In the case of Bad-food.owl, we report two issues. Firstly, the times taken for this ontology were similar in all three tools; the subjects in Group B took relatively less time to debug this ontology than when they were debugging Mad_Cow.owl. Secondly, the explanation function of OWLDebugger was rated to be more useful than RepairTab and SWOOP. The following is our explanation for this observation. OWLDebugger explains the error was due to the disjoint axiom, and hence some subjects immediately chose to remove this axiom without analysing the cause of the unsatisfiability. We noticed that, from the survey they filled, three subjects did not understand the cause of the error, even though they resolved the error successfully. On the other hand, two subjects using RepairTab found it difficult to analyse the problematic axioms which were presented in the formal DL notations. This problem was pronounced with Bad-food.owl because the axioms are relatively complicated. Furthermore, the fine-grained approach was not applicable because all parts of the axioms are responsible for the unsatisfiability, and hence all were highlighted as problematic. This explanation given for this example is similar in SWOOP. As a result, the subjects using RepairTab or SWOOP found it difficult to understand the reason for the unsatisfiability and to decide which changes should be made.

For the University.owl ontology, we report two issues. Firstly, the rating of usefulness of the explanation in RepairTab and OWLDebugger is higher than that of SWOOP. This is because, as mentioned before, for the unsatisfiable concept Person, SWOOP strikes out the whole right-hand side of a problematic axiom; some subjects thought that the explanation was confusing. Secondly, the subjects using RepairTab or SWOOP took less time to complete the task than those using OWLDebugger. We suggest the following two reasons: (1) RepairTab sorted the twelve unsatisfiable concepts in order of size of problematic axioms. The subjects were guided to debug the concept with the least number of problematic axioms first. SWOOP highlights the root and derived unsatisfiable concepts. When a concept was resolved, most of its subconcepts were resolved as well. However, the subjects in Group B had to explore each unsatisfiable concept one by one. (2) RepairTab and SWOOP provide previews of the impact of removal, but OWLDebugger does not. We noticed that two subjects in Group B simply completed the task by removing the disjoint axioms, without understanding the reasons behind the unsatisfiabilities; the subjects did not realise that these removals caused many lost entailments. On the other hand, we noticed that the subjects using RepairTab or SWOOP were discouraged from this type of removal because they could preview the impact of removal. Three subjects in Group A revised their changes after exploring the consequences of different modifications (i.e., after seeing many lost entailments or more helpful changes provided). For the subjects using SWOOP, they tried to remove some axioms and preview the impact on the ontology. When the subjects removed an axiom, they checked how many fixed and remaining unsatisfiable concepts, as well as the lost and retained entailments occurred. Some subjects chose to remove axioms which caused fewer lost entailments and more retained entailments. The repairing functionality helped them to debug ontologies with many unsatisfiable concepts. However, the lost entailments facility could be improved; in some cases, the lost entailment is exactly the same as the axiom just removed by the subjects, and hence not that helpful. Furthermore, there is no functionality to add the lost entailments back to the ontology. Therefore, we claim that RepairTab has some advantages; it is able to minimise the impact on the ontologies in the case of removing (parts of) axioms, by providing the helpful changes facility.

Interestingly, we found that some subjects claimed they understood the reasons for the unsatisfiability, but they simply deleted disjoint axioms or subclass-of relationships, particularly in the University.owl ontology. Therefore, we classified these subjects as not understanding the errors. We believe this ontology, which contains one of the most common OWL modelling errors, is very difficult for the subjects. In the case of Person in University.owl, none of the subjects realised that $\mathsf{FrenchUniversity} \doteq \forall\, \mathsf{offerCourse.Frenchcourse}$, the domain of offersCourse is University, FrenchUniversity is a subclass of University, therefore, University is defined as equivalent to owl:Thing implicitly, then Person which is disjoint with owl:Thing is unsatisfiable. Two subjects using SWOOP did not realise that SWOOP displays the implicit axiom $\mathsf{University} \doteq \mathsf{owl{:}Thing}$, they removed the domain or disjoint axiom to resolve the problem. However, when some subjects in Group A were exploring the removal of $\mathsf{FrenchUniversity} \doteq \forall \mathsf{offerCourse.Frenchcourse}$, they discovered that a helpful change $\mathsf{FrenchUniversity} \sqsubseteq \forall \mathsf{offerCourse.Frenchcourse}$ could be added back to the ontology, and they decided to make this change.

8.3.1.2 Summary of the Usability Study

In general, the subjects in the study appreciated the debugging functionalities provided by the three tools. The subjects were guided on where to start debugging, and the explanations provided by the tools helped them to understand the cause of errors. Both RepairTab and SWOOP provide functionalities to resolve problematic axioms; they allow the subjects to preview the impact of removal on the ontologies before the removal is committed. However this function is limited to *remove (parts of)* axioms, no support for *changing* parts of axioms is provided. As a result, many subjects just removed (parts of) axioms to resolve the errors. In contrast, OWLDebugger does not provide a removal function, because of this some subjects modified the definitions of concepts instead of removing axioms directly.

SWOOP [91] provides a number of debugging services, which are: the axiom pinpointing service[14], the root/dervied error pinpointing service[15] and the ontology repair service[16]. The implementors of SWOOP conducted a number of usability studies to evaluate the usefulness of these services by giving their subjects different debugging services. For example, their results showed that the information provided by the axiom pinpointing service is better than no support for debugging ontology, and that the ontology repair service is more effective than both the axiom pinpointing and the root/derived error pinpointing service. However, there is no study to compare SWOOP with other ontology debuggers such as OWLDebugger. From our study, we further revealed the advantages and limitations of SWOOP by comparing with OWLDebugger and RepairTab. For example, the function 'striking out irrelevant part of axioms' is incomplete, the axioms presented in DL notations are difficult for the subjects to understand the cause of the unsatisfiability, and the explanations for the lost and retained entailments due to the removal of axioms are useful. The advantages and disadvantage of the three tools are further discussed in Section 8.4.

8.3.2 Performance Analysis

The non-determinism in the expansion rule (i.e., $\sqcup$-rule) results in poor performance of the tableau algorithm. Existing DL reasoners, e.g., FaCT++ [146], Racer [60] and Pellet [136], employ optimisation techniques. They have demonstrated that even with expressive DLs, highly optimised implementations can provide acceptable performance in realistic DL applications. For example, dependency directed backtracking is used to prune the search tree [74]. However, in our fine-grained approach, these techniques are no longer applicable, as we aim to detect all possible clashes by fully expanding the tree. This will adversely affect the performance of the algorithm especially if there is extensive non-determinism in the ontology. Considering that the number of unsatisfiable concepts is relatively small compared to the total number of concepts in realistic ontologies, we believe it is practical to first check the consistency of ontologies using optimised

[14]The axiom pinpointing service is to pinpoint the axioms which cause a concept to be unsatisfiable.

[15]The root/dervied error pinpointing service is to distinguish the root unsatisfiable concepts and the derived ones.

[16]The ontology repair service is to rank erroneous axioms and provide rewriting axiom suggestions.

reasoners to find the unsatisfiable concepts, and then run the fine-grained algorithm on those unsatisfiable concepts. For example, Sweet-JPL.owl[17] contains 1537 concepts, and one concept is unsatisfiable. Hence, our algorithm is then only applied to the unsatisfiable concept, instead of all of the concepts in the ontology.

In this subsection we reported on two types of benchmark experiments: we conducted some experiments with our own built data-set, and an evaluation with a number of realistic ontologies. RepairTab was implemented in Java. The tests were performed on a PC (Intel Pentium IV with 2.4GHz and 1GB RAM) with Windows XP SP2 as operating system. Firstly, we adopted the method described in [133] to construct unsatisfiable $\mathcal{ALC}$ concepts (disjunctions are of size 14 on average). We varied the number of unsatisfiable concepts from 1 to 100 in the test, and then compare the results with an optimised reasoner. The reason for focusing on the number of unsatisfiable concepts is because our approach uses an optimised reasoner to trim away all satisfiable parts of the ontology in advance, as described above. Figure 8.10 shows the results of RepairTab and Pellet. As shown in the graph, the execution time of RepairTab dramatically increases with the number of unsatisfiable concepts. Note that the benefit of our approach is to obtain all possible clashes which cause a concept to be unsatisfiable, by fully expanding the tree. In comparison, Pellet performs very well, as optimisations such as early clash detection, dependency directed backjumping and semantic branching are used. Also Pellet only tests if concepts are unsatisfiable, but does not detect all possible clashes. An alternative to fully expand the tree is to use Reiter's Hitting Set Tree approach coupled with removing axiom parts in order to find all MUPSs (see [87], Section 4), this is discussed in Section 4.4.

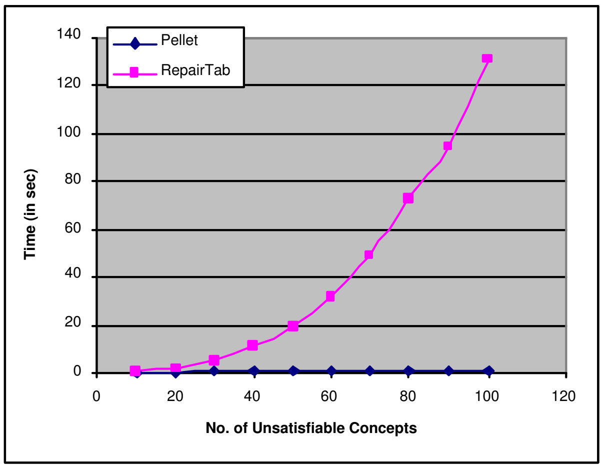


Figure 8.10: Performance Comparison of Pellet and RepairTab for Satisfiability Checking

Secondly, benchmarking with real-life ontologies is obviously a convincing way to evaluate the quality of our approach. However, there is only a limited number of realistic ontologies that are both represented in $\mathcal{ALC}$ and unsatisfiable. We therefore constructed simplified $\mathcal{ALC}$ versions for a number of ontologies downloaded from the Internet. We then removed, for example, numerical constraints, role hierarchies and instance information. As some ontologies are satisfiable, we randomly changed them such that each change on its own leads to unsatisfiable concepts. For

[17]http://www.mindswap.org/ontologies/debugging/buggy-sweet-jpl.owl

Ontology	♯ of Unsat. Concepts	RepairTab	Pellet	% Incr.
University[a]	12	0.147	0.125	0.176
Bad-food[b]	1	0.057	0.047	0.212
Koala[c]	3	0.093	0.078	0.192
Sweet-JPL[d]	1	0.327	0.312	0.048
Mad_cow[e]	1	0.105	0.094	0.12
Economy[f]	30	0.425	0.359	0.184
Transportation[g]	20	0.552	0.5	0.104

[a]http://www.mindswap.org/ontologies/debugging/university.owl
[b]http://www.mindswap.org/dav/ontologies/commonsense/food/foodswap.owl
[c]http://protege.stanford.edu/plugins/owl/owl-library/koala.owl
[d]http://www.mindswap.org/ontologies/debugging/buggy-sweet-jpl.owl
[e]http://cohse.semanticweb.org/ontologies/people
[f]http://reliant.teknowledge.com/DAML/Economy.owl
[g]http://reliant.teknowledge.com/DAML/Transportation.owl

Table 8.5: Results of Testing Existing Ontoloiges (Times are shown in seconds)

example, we added disjointness statements among sibling concepts, and introduced some common ontology modelling errors enumerated by Rector et al. [125]. Figure 8.5 shows the average runtime (in seconds) of the satisfiability test of a set of ontologies. The execution time of our extended algorithm is increased by 15% on average compared with that of Pellet, because our algorithm aims to detect all possible clashes given that it requires a fully expanded tableau tree, while many optimisations are disabled. In the cases of Transportation.owl and Economy.owl, the running time for checking the satisfiability of 20 and 30 concepts is less than 0.6 seconds. The result shows that the performance of our approach is feasible in realistic ontologies which do not contain a large number of unsatisfiable concepts.

We were particular interested in the GALEN ontology[18], which models medical terms and procedures. It contains over 2700 classes and 400 GCIs. As its DL expressivity is $\mathcal{SH}f$, we constructed a simplified $\mathcal{ALC}$ version of it. Figure 8.11 shows the average runtime (in seconds) from 1 to 1000 satisfiability tests. The execution time of RepairTab dramatically increases with the number of unsatisfiable concepts for the reason given above. Note that there is a large number of GCIs in the GALEN ontology, the optimised reasoner is able to eliminate non-determinism by absorbing them into primitive concept introduction axioms whenever possible. In our approach, this technique is still applicable, because it is algorithm independent. Section 4.3.2.1 provides more details.

8.4 Comparison of Debugging Tools

From the results of the usability evaluation, the tools OWLDebugger, SWOOP and RepairTab have been shown to be generally useful for the subjects to debug ontologies. We now compare the disadvantages and advantages of the tools in two aspects: usability and functionality.

[18]http://www.cs.man.ac.uk/∼horrocks/OWL/Ontologies/galen.owl

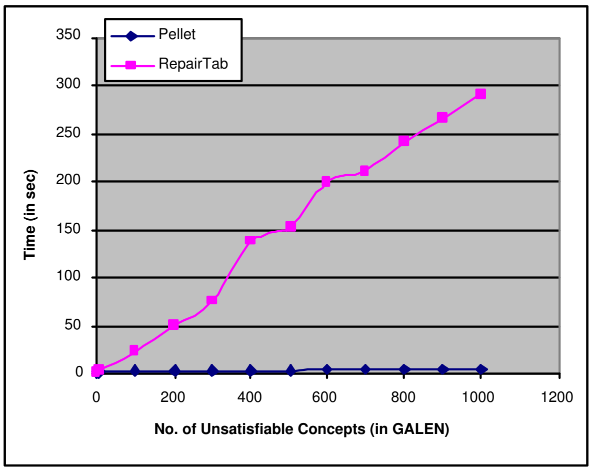


Figure 8.11: Performance Test on Pellet and RepairTab in the GALEN ontology

8.4.1 Usability

As we mentioned before, many ontology users find it difficult to understand the logical meaning and potential inferences of statements in description logics, including OWL-DL [125]. Therefore, it is important to explain the cause of unsatisfiable concepts in a way that is understandable to ontology users. An explanation screenshot of the three tools is shown in Figure 8.12. We compare the usability of these tools in Table 8.6.

We conclude that RepairTab highlighting the problematic parts of axioms is useful, its usability still has room for improvement. Some subjects in the usability study suggested that it would be more user friendly if RepairTab used the Manchester OWL Syntax [73] in which description logic symbols such as $\exists$, $\forall$, $\neg$ are replaced by keywords such as “some”, “only” and “not”. To further help non-expert users to debug ontologies, natural language techniques [20; 69; 111; 156] can be adopted to explain inconsistencies in ontologies.

8.4.2 Functionalities

The three tools provide different support for users to debug ontologies. In Table 8.7, we compare the functionalities of ranking and rewriting axioms in SWOOP and RepairTab. OWLDebugger is not considered, as it does not support ranking or rewriting axioms.

From the comments of the subjects using RepairTab, the main limitation of RepairTab is that it only supports the removal of parts of axioms. However, in some cases, the desired changes are not removal, but changes to parts of axioms. A similar problem arose with SWOOP whose repair service only allows users to remove whole axioms.

8.4.3 Summary

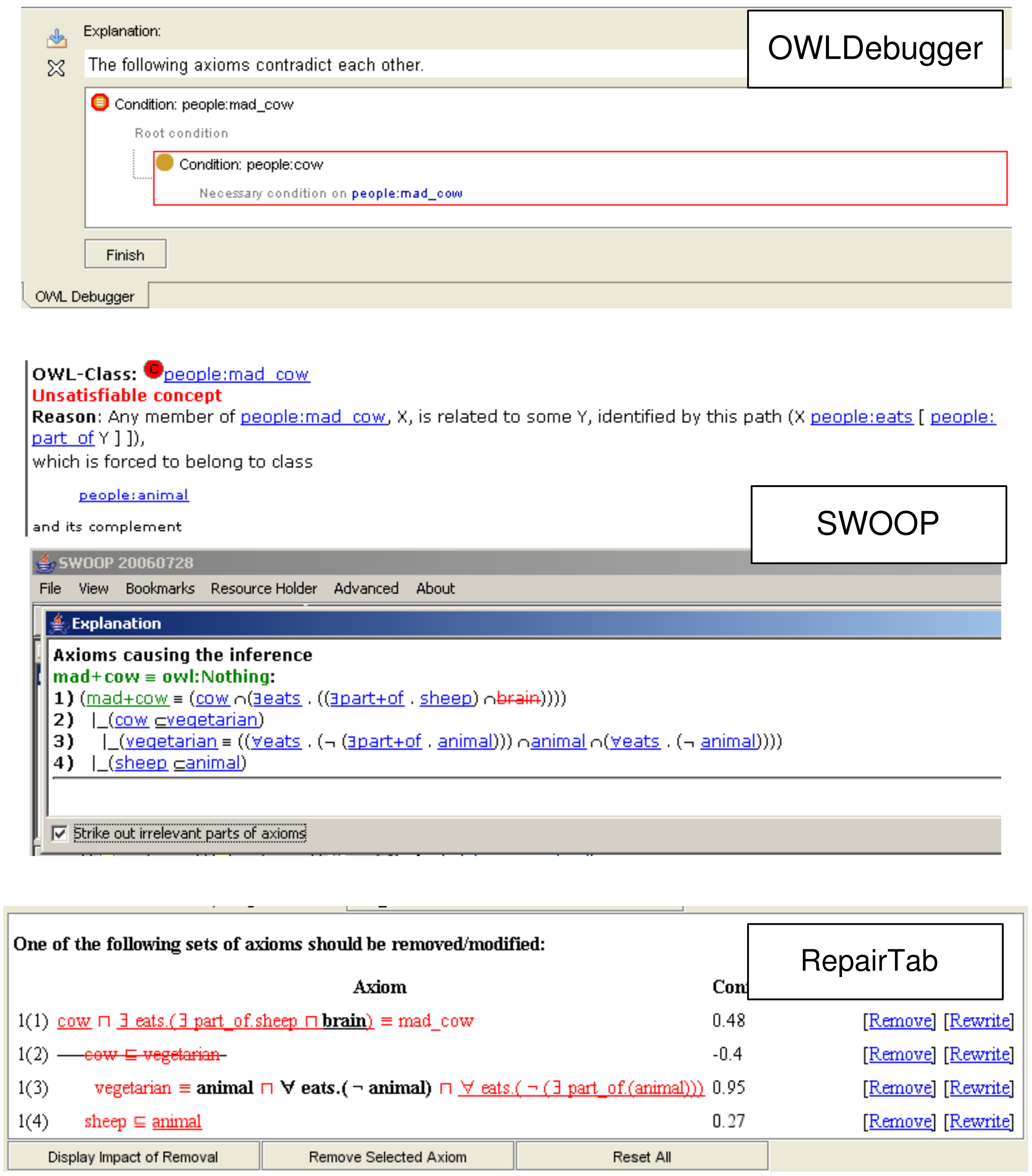
Currently, we cannot conclude which tool is the best for debugging ontologies. The subjects in the study each used different tools for debugging, hence they could not compare the tools and rate them relative to one another. Furthermore, the numbers of subjects and ontologies are too small to provide a recommendation for which tool might be the best on a larger set of ontologies. We

Tool	**Advantages**	**Disadvantages**
OWLDebugger	(1) Quasi-natural language explanations are generated, they are easy to understand. (2) The debugger leads the user through various debugging steps, and provides the user with suggestions which classes should be debugged.	(1) Explanations are oversimplistic when the cause of the unsatisfiability is too complex to explain. This is because it uses a set of heuristics rules to generate explanations. (See Section 3.3.2 for more details.) (2) It does not indicate the problematic parts of axioms.
SWOOP	(1) It displays the reason for a clash in an unsatisfiable concept C, that any member of C is forced to belong to class C and its complement. (2) It hightlights the root and derived unsatisfiable concepts.	(1) The displayed clash information of an unsatisfiable concept, in some cases, is too simple to explain errors caused by complex axioms, as SWOOP only points to the 'last' contradictory part in the tableau tree. (2) The function 'striking out irrelevant parts of axioms' does not always work. (3) The axioms are displayed in formal DL notations. Some subjects in the usability study commented that they found it difficult to understand some of the more complex axioms.
RepairTab	(1) It highlights the parts of axioms which cause the unsatisfiability. (2) It sorts the unsatisfiable concepts in order of size of problematic axioms.	(1) It does not display clash information for unsatisfiable concepts. (2) It displays axioms in DL notations which is difficult for some users to understand.

Table 8.6: Comparison of the usability of OWLDebugger, SWOOP and RepairTab

see two possible ways to evaluate the relative usefulness of the tools. Firstly we could choose to use an objective measure such as the time taken by subjects for debugging and the quality of resulting ontologies. In this first case we could have different (randomly selected) groups of users using each tool. The second alternative would be to have a group of users using each tool in turn (the order could be randomised); these users could then give their subjective ratings of the relative merits of each tool. For this experiment to be effective all the subjects would need to have detailed experiences of all three tools.

We learnt some useful lessons based on the results of our usability studies, and the feedback given by the subjects. As discussed in Sections 1.2 and 2.4.1, the support for explaining unsatisfiability is very important for newcomers to OWL-DL ontologies, including for those experts with a strong background in databases or object-oriented systems. The first step to assist users to debug ontologies is to explain the inconsistencies in easily understandable ways. We understand that translating the DL axioms into natural language, or displaying the axioms using a graphic user interface would be very useful for the users. Moreover, it would be helpful to explain the implicit information which causes unsatisfiability, instead of just displaying all problematic axioms.



Explanation:

The following axioms contradict each other.

Condition: people:mad_cow

Root condition

Condition: people:cow

Necessary condition on people:mad_cow

OWL-Class: people:mad_cow

Unsatisfiable concept

Reason: Any member of people:mad_cow, X, is related to some Y, identified by this path (X people:eats [people:part_of Y]]),
which is forced to belong to class

people:animal

and its complement

Axioms causing the inference

mad+cow ≡ owl:Nothing:

1) (mad+cow ≡ (cow ∩(∃eats . ((∃part+of . sheep) ∩~~brain~~))))

2) |_(cow ⊂vegetarian)

3) |_(vegetarian ≡ ((∀eats . (¬ (∃part+of . animal))) ∩animal ∩(∀eats . (¬ animal))))

4) |_(sheep ⊂animal)

☑ Strike out irrelevant parts of axioms

One of the following sets of axioms should be removed/modified:

	Axiom	**Con**	
1(1)	cow ⊓ ∃ eats.(∃ part_of.sheep ⊓ **brain**) ≡ mad_cow	0.48	[Remove] [Rewrite]
1(2)	~~cow ⊑ vegetarian~~	-0.4	[Remove] [Rewrite]
1(3)	vegetarian ≡ **animal** ⊓ **∀ eats.(¬ animal)** ⊓ ∀ eats.(¬ (∃ part_of.(animal)))	0.95	[Remove] [Rewrite]
1(4)	sheep ⊑ animal	0.27	[Remove] [Rewrite]

Figure 8.12: Explanations Screenshot of the Debugging Tools

Example 43 *For the koala.owl example*[19], Koala *is unsatisfiable due to the following axioms:*

1. Koala ⊑ Marsupials

2. Koala ⊑ ∃ isHardWorking.*"false"*⟨ *xsd:boolean*⟩

3. domain(isHardWorking) = Person

4. Person ⊑ ¬Marsupials

Both SWOOP and RepairTab display the above problematic axioms in DL notations for explanation. Note that it is a common problem for users to misunderstand domain axioms in OWL-DL [125], they might not notice that Koala is a subclass of Person implicitly due to axioms 2 and 3. Therefore, a debugging tool should explain the inferred information explicitly, and hence, the user is guided to focus on axioms 2 and 3 to resolve the error, instead of all of the axioms.

[19]http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

Tool	**Advantages**	**Disadvantages**
	Ranking Axioms	
SWOOP	(1) Axioms are ranked in order of importance which is dependent on arity, impact and usage. (2) The ranking of axioms is explained by dispalying the resource for the numerical arity, impact and usage values	(1) Ranking strategies are quite limited, for example humans' heuristics are not considered. (2) Ranking strategies are not distinguished for ontologies containing common errors and unsatisfiable ontologies merged from satisfiable ontologies.
RepairTab	Axioms are evaluated with confidence value based on two sets of heuristics, namely, common error heuristics and merging heuristics	Axioms are just assigned with a numeric confidence value, but no explanation for the value is provided.
	Removing/Modifying Axioms	
SWOOP	(1) After removing an axiom, the fixed and remaining unsatisfiable concepts, as well as the lost and retained entailments are listed. (2) In some cases, `(Why?)` hyperlinks are provided to explain the lost and retained entailments. (3) It is possible to preview the impact on the ontology before the removal is committed.	(1) Only 'removal of whole axioms' is provided. It does not provide support for removing or changing parts of axioms. (2) When an axiom is removed, it does not explain why some unsatisfiable concepts are fixed, why others remain unsatisfiable. (3) Lost entailments are limited to axioms of the form $\mathsf{CN} \sqsubseteq \mathsf{DN}$ or $\mathsf{CN} \sqsubseteq \neg \mathsf{DN}$ (where CN and DN are atomic concepts).
RepairTab	(1) It allows the user to add helpful changes back into the ontology. (2) It allows the users to remove *parts of* axioms and preview the impact of removal, they are not limited to removing whole axioms. (3) When (parts of) axioms are removed, the lost entailments are not limited to axioms of the form $\mathsf{CN} \sqsubseteq \mathsf{DN}$ or $\mathsf{CN} \sqsubseteq \neg \mathsf{DN}$. (see Section 7.2)	(1) It only provides support for the removal of (parts of) axioms, there is no support for changing parts of axioms. (2) It does not provide any explanations for why the lost entailments are lost, nor for helpful and harmful changes.

Table 8.7: Comparison of Ranking and Rewriting Axioms in SWOOP and RepairTab

After understanding the reasons for unsatisfiability, the next step is to rewrite axioms, rather than remove them directly from the ontology. The subjects in our studies claimed that they understood the cause of the errors, but they did not know how to correct them, and so rewriting suggestions would be helpful. We will further discuss rewriting axioms suggestions in Section 9.4.2.

Chapter 9

Conclusion

This thesis concludes with a review of the work presented and an assessment of the extent to which the objectives set out in Chapter 1 have been met. The significance of the major results is summarised, outstanding issues are discussed and, directions for future work are suggested.

9.1 Thesis Overview

The objective of this thesis was to investigate methods for supporting ontology modellers to resolve inconsistencies in ontologies. A fine-grained approach was proposed to identify problematic parts of axioms. In order to resolve the unsatisfiable concepts, the user has to select appropriate axiom(s) to be removed/modified. Our heuristic-based service utilised a variety of heuristics to recommend the user to select appropriate axiom(s) to remove/modify. In the case of removal, we aim to reduce the impact of changes on the ontology. The fine-grained approach was used (1) to calculate lost entailments of named concepts due to removal of (parts of) of axioms, and (2) to identify changes which are helpful in that they restore lost entailments, and those changes that are harmful in that they cause additional unsatisfiability. A caching technique is proposed to improve the efficiency of consistency checking for modified ontologies.

The practical usefulness of these methods was demonstrated by several usability studies. The results of the studies showed that the heuristic-based service and the fine-grained approach were useful for the non-expert users to resolve inconsistencies in ontologies, and minimise the impact of changes on the ontologies.

9.2 Significance of Major Results

The main contributions of this thesis are:

[**1**] A *fine-grained approach* identifies the problematic parts of axioms, which are responsible for an unsatisfiable named concept.

[**2**] A variety of *human heuristics* were acquired from our empirical studies, in which the subjects were asked to think-aloud when resolving inconsistencies in ontologies.

[**3**] A *heuristic-based service* is proposed to recommend the users to select appropriate problematic axioms for removal or modification.

[**4**] We utilised the fine-grained approach to calculate detailed information about *the impact of changes* on the ontology. Specifically, we are able to identify changes which are *helpful* in that they restore some lost entailments (due to axiom removal), and those that are *harmful* in that they cause additional unsatisfiability.

[**5**] A *caching technique* is proposed to improve the efficiency of consistency checking for modified ontologies.

Fine-grained Approach

The fine-grained approach allows us to rewrite faulty axioms by removing error-causing concepts, instead of removing complete axioms (cf. Chapter 4). The result of the usability study in Section 8.3.1 shows that the highlighted parts of axioms helped the subjects to analyse the cause of concepts' unsatisfiability. Comparing with other debugging tools, namely, OWLDebugger and SWOOP, this function was rated to be more useful, except that, in some cases, all parts of the axioms are responsible for the unsatisfiability, and hence the whole axioms are highlighted, and so the fine-grained approach is not helpful for explanation. We also learnt from the subjects of the study that, to further improve the usability, the DL syntax used by RepairTab could be difficult for non-expert users to understand, and quasi-natural language explanation would be useful.

As the subsumption test can be reduced to the satisfiability test [134], this approach is also applicable for identifying the parts of axioms which are responsible for a subsumption relation between two named concepts.

Human's Heuristics

A variety of *human heuristics* were acquired from our empirical studies, in which the subjects were asked to think-aloud when resolving inconsistencies in ontologies. We found that some of our acquired heuristics were already known and have already been incorporated in existing tools, but we have also discovered new useful heuristics (cf. Chapter 5). Knowledge of these heuristics could be invaluable to those who build ontology debugging tools to help users resolve inconsistencies. Interestingly, our studies also revealed some contradictory heuristics. We also gained insight into ways in which apparently contradictory heuristics could be made useful if provenance information or information about the process by which the ontology was constructed is also available.

Heuristic-based Service

After identifying minimal sets of problematic axioms which are responsible for a concept's unsatisfiability, at least one axiom has to be removed/modified from each set. Selecting appropriate axiom(s) for removal/modification might be difficult for non-expert users, we therefore proposed a *heuristic-based service* to recommend the users to make such selections. We firstly acquired a number of heuristics from our empirical studies and adapted some heuristics from the literature, and then formalised the heuristics which allow us to evaluate the *confidence* of axioms.

These heuristics are incorporated into an ontology debugging tool, "RepairTab". From the results in the empirical studies (reported in Chapter 5), we learnt that some subjects (the ontology experts) took common OWL modelling errors into account to resolve the inconsistencies in ontologies; some subjects took the process of constructing ontologies into account, such as integrating/merging ontologies. We therefore, clustered the heuristics into two sets: "common error heuristics" and "merging heuristics", then conducted a usability study with non-expert users. The results of our usability study (cf. Section 8.2.3) show that (1) with the heuristic support, the users were guided towards the axioms which would be the best to modify, and they achieved correct modifications in a shorter time; (2) the heuristics could increase the users' confidence in the modifications which they are making; (3) the common error heuristics were useful for dealing with ontologies with common mistakes; (4) the merging heuristics were useful for dealing with merged ontologies. The heuristic approach is an important starting point in ontology debugging. Similar techniques could be applied to resolve inconsistencies arising from ontology learning, ontology mapping, or migration to OWL (e.g., Tambis.owl and DICE.owl ontologies).

Impact of Removal

In the case of removing (parts of) axioms, it frequently happens that rewriting axioms might not resolve the unsatisfiable concepts, but might introduce unintended lost entailments, and such entailments can be far from obvious. In order to *minimise the impact of changes* and prevent unintended entailment loss, we proposed *helpful* and *harmful* changes, those which are helpful restore lost entailments (due to axiom removal), and those that are harmful cause additional unsatisfiability. The results of the usability study (cf. Section 8.3) show that (1) the information of the loss of entailments due to removing (parts of) axioms is helpful for users to minimise the impact of changes on the ontology; and (2) the helpful changes facility is helpful to prevent unintended/implicit entailment loss. For example, in the usability study, after previewing the impact of changes on the ontology, some subjects were discouraged from removing some axioms whose removal would cause many lost entailments. Also some subjects using RepairTab tried to remove parts of an axiom, they were surprised to see that some helpful changes could be added back to the ontology.

As we aim to detect all possible clashes by fully expanding the tableau tree, this will adversely affect the performance of the algorithm especially if there is extensive non-determinism in the ontology. We therefore first check the consistency of ontologies using an optimised reasoner to find the unsatisfiable concepts, and then run the fine-grained algorithm on those unsatisfiable concepts. The performance evaluation in Section 8.3.2 shows that our approach is feasible in realistic ontologies which do not contain a large number of unsatisfiable concepts. Furthermore, with the fine-grained approach, during the removal of (parts of) axioms, we are able to calculate the lost entailments directly from the sequences of clashes (which are obtained from the tableau tree), rather than running the reasoner to detect the lost entailments due to removal.

These techniques are also applicable to calculate the impact of updating axioms on satisfiable/consistent ontologies, or the impact of partitioning ontologies. For example, if an ontology is partitioned into two sub-ontologies, it would be useful to calculate the information lost in the sub-ontologies compared to the original one.

Caching Technique

Lastly, the proposed helpful and harmful changes are calculated with respect to an unsatisfiable concept. To know if a change is harmful to other satisfiable named concepts, or helpful to other unsatisfiable named concepts, we have to re-check the consistency of the whole ontology by running the tableau algorithm. In order to increase the efficiency of satisfiability checking when updating axioms, we make use of the cached results from previous runs of the tableau algorithm to check the (un)satisfiability of concepts, without running the tableau algorithm from scratch. This approach is particularly useful for updating large ontologies, as a considerable amount of work can be saved.

9.3 Outstanding Issues

This thesis does not, of course, represent a complete answer to the problems of resolving unsatisfiable ontologies. There remain a sizeable number of outstanding issues, several of which are discussed below.

9.3.1 Fine-grained Approach

As discussed in Section 4.3, the fine-grained approach has limitations on the expressivity of DLs and optimisations.

More Expressive DLs Currently, the fine-grained approach only supports terminologies in $\mathcal{ALC}$ with GCIs. In order to increase the applicability of the approach, it is necessary to extend the algorithm to support the Description Logic $\mathcal{SHOIN}(\mathbf{D})$ [79] as an OWL-DL ontology corresponds to a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base.

Optimisations Given an unsatisfiable concept, we aim to detect all possible clashes in the tableau tree, and the fine-grained algorithm we use requires us to fully expand that tree. This adversely affects the performance of the algorithm when there is extensive non-determinism in the ontology axioms, by way of disjunctions (in $\mathcal{ALC}$). Note that existing DL reasoners, such as FaCT++ [146], RACER [60] and Pellet [136], employ optimisation techniques. They have demonstrated that even with expressive DLs, highly optimised implementations can provide acceptable performance in realistic DL applications. Most importantly, lazy unfolding, GCI absorption and caching are still applicable in our algorithm. Other optimisations, such as dependency-directed backtracking and semantic branching search, increase efficiency by avoiding fully expanding the tree, and so they are not compatible with our algorithm. We need to further investigate how to revise our algorithm to adopt these optimisations.

9.3.2 Heuristic-based Approach

Currently, we cluster the heuristics into two sets: ‘common error heuristics’ and ‘merging heuristics’. In the usability study, a small number of subjects and ontologies were used. Also, the subjects using heuristics from SWOOP and the subjects using our ‘common error heuristics’ had similar performance. Based on our results, we cannot determine if our 'common error heuristics' is more useful than SWOOP's heuristics. In the future work, the following two options could be pursued: (1) evaluate each heuristic individually; (2) evaluate different combinations of heuristics, different weights on heuristics, or different types of ontologies (such as domain ontologies, application ontologies, or task ontologies). As mentioned in Section 8.2.4.1, to implement such evaluations, we need a very large number of unsatisfiable ontologies for a complete study. It might not be practical to carry out such a huge study. Instead, when there are more unsatisfiable ontologies available on the Web in the future, we would create different combinations of heuristics (include those described in Section 6.3.10), weights of heuristics and types of ontologies, in order to evaluate the interactions between heuristics in terms of the types of ontologies, different sets of heuristics, or the weights of heuristics.

9.3.3 Usability Issues

In our usability study, some subjects using RepairTab or SWOOP commented that the presentation of DL notation made it difficult for them to understand the meaning of complex concept definitions. Even though our tool identifies the faulty parts of axioms to explain the unsatisfiability of the concepts, some subjects failed to understand the logical formalism. On the other hand, subjects using OWLDebugger appreciated that the quasi-natural language explanation was very useful for them to understand the cause of the contradictory concepts. In order to further support users debugging ontologies, we could adopt the Manchester OWL Syntax [73] in which some mathematical symbols such as $\exists$, $\forall$, $\neg$ are replaced by keywords such as "some", "only" and "not". Also, the generation of natural language (NL) paraphrases for DL axioms based on a variety of NLP techniques [20; 69; 111; 156] would be useful.

Moreover, the explanation of the confidence value of axioms generated by the heuristics would be very useful, so that the user could understand the reasons for the displayed confidence values of axioms. For example, given a disjoint axiom with a high confidence value, the tool could explain to the user that it is good practice to usually have disjoint sibling concepts; for a disjoint axiom with a low confidence value, the tool could explain to the user that it is unusual for two named concepts which are far from their common ancestor to be disjoint.

Since all concepts are normalised in the tableau algorithm, the generated harmful and helpful changes are in negation normal form. That means that the format of displayed changes of axioms might be different from the original ones. In addition, recall that absorption is used to preprocess the ontology before the tableau algorithm is applied; because of this we can only identify the parts of the axioms (modified by the absorption) responsible for an unsatisfiability, instead of the originally asserted axioms in the ontology. To further improve the usability, our future work will involve displaying the information in its original format. Lastly, it would be useful to explain why the suggested "helpful changes" are helpful, and why the "harmful changes" are harmful.

9.4 Future Work

The heuristic-based ranking of axioms and the fine-grained approach for supporting rewriting axioms have opened several promising paths for further research. Some of them are presented below.

9.4.1 Heuristic-based Approach

9.4.1.1 Heuristics for Ontology Mapping, Learning and Migration

We believe that the results of our usability study are an important starting point in ontology debugging. We notice that inconsistencies in ontologies are common in ontology engineering. Firstly, in ontology mapping [119], inconsistent ontologies might result from joining pairs of ontologies using automatically generated mappings. As discussed in Section 8.2.1, some ontology mapping systems (such as COMA++ [108], Falcon-AO [82] and HMatch [26]) create the union of two ontologies by inserting additional (and sometimes incorrect) axioms. For example, given two concept names, ConferencePaper and ConferenceMember, COMA++ adds a new subsumption relation between them using lexical similarity in the resulting ontology, however, the new axiom might cause contradictions. Secondly, in ontology learning [31], inconsistent ontologies might result from constructing ontologies semi-automatically by learning from text, from a dictionary, a knowledge base, or a semi-structured data source. Thirdly, in migrating some data sources (e.g., frame-based knowledge source) to OWL ontologies, inconsistencies might occur due to the different formalisms. For example, the OWL version of the Tambis ontology contains 144 unsatisfiable classes (out of 395) due to an error in the transformation script used in the conversion process [86]. Further, the DICE terminology (a DL terminology in the Intensive Care domain) suffered from a high number of unsatisfiable concepts due to its migration from a frame-based system [131].

We believe that it is useful to conduct similar empirical studies to acquire humans' heuristics to cope with inconsistencies in ontologies which result from the above causes.

9.4.1.2 Provenance Information

We learned from the empirical studies that when the subjects chose which axioms should be removed by evaluating the *correctness* and *importance* of axioms in the ontologies, their heuristics could be somewhat contradictory. We have gained insight into ways in which apparently contradictory heuristics could be made useful if provenance information (e.g., reliability of the sources or the level of expertise of the modellers) or information about the process by which the ontology is constructed can also be made available. For example, if an ontology management system has access to information about the level of expertise of the user, then the tool could provide heuristics which are useful for common errors in ontologies for a novice. If the tool has access to information about the process through which the ontology is constructed, or the reliability of the sources of the ontology, then the axioms from reliable sources could be assigned higher confidences, otherwise, the axioms asserted by the modeller could be assigned higher confidence. Also, it is common for users to extend/reuse existing ontologies. It is reasonable to assign higher confidences to the local axioms over axioms from imported ontologies. With the owl:imports construct, we cannot import part of an ontology and leave out other parts [55]. Therefore, imported axioms which contradict existing ones have low reliability and, hence, we have lower confidence in them. In collaborative ontology building scenarios, different modellers can be assigned with precedence levels representing the status or authority of the user, and hence axioms added by a user with a higher precedence level will be given higher importance [98; 91].

9.4.1.3 Customisation and Sharing of Heuristics

Currently, the heuristics are “hard-wired” in the tool. This means that adding new heuristics requires programming. The following two approaches could address this limitation:

Customisation of Heuristics The heuristics can be customised by the user or set to default values by the system, in which case the ranking of axioms may have different weights. Firstly, users can assign different weights to heuristics to reflect their preferences. Secondly, the system can log the actions performed by the user into log files, and evaluate how often the heuristics have been used. Therefore, the system is able to customise the heuristics according to changes actually made on an ontology by a particular user.

Sharing of Heuristics The system should support import and export of heuristics. The heuristics can be kept in a library stored in a shared folder, where the contexts of ontologies are also retained. Therefore, the user can share other users’ preferences for heuristics; the user can also explore how other users customise their heuristics. Furthermore, the system can adjust the weight of heuristics based on the popularity of heuristics among users.

9.4.2 Rewriting Axioms Suggestions

From the empirical studies described in Chapter 5, we observed that the subjects had a number of strategies for rewriting axioms, such as splitting classes, generalisation (move a class to a higher level in the hierarchy, change intersection to union, change cardinality values. etc.). Furthermore, the subjects in our usability evaluation also suggested that it would be very helpful to provide rewriting suggestions, in addition to ranking axioms. It will be worthwhile to analyse the following three strategies for rewriting axioms.

9.4.2.1 Methods for Weakening Restrictions

There are many options for weakening restrictions to resolve inconsistencies, such as changing to a higher level class of a hierarchy, weakening cardinality restrictions. Plessers [122] propose a set of rules to resolve the detected inconsistency. They argue that an ontology is logically inconsistent because the axioms of the ontology which are contradicting each other are too restrictive. To resolve the inconsistency, the restrictions imposed by the axioms should be weakened. They weaken restrictions either by removing an axiom, replacing it with its superconcepts, or changing its cardinality restriction values.

9.4.2.2 Strategies derived from Common Errors

As described in Section 3.3.2, Rector et al. [124] identify the common problems of ontology users during modelling OWL ontologies, and provide a summary of guidelines. Kalyanpur et al. [91] further extend the list of the common errors, and keep a library of common errors in modelling OWL ontologies. Kalyanpur et al. [91] check if any of the axioms has a pattern corresponding to the axioms in the library, and if so, they suggest the intended axiom to the user as a replacement.

9.4.2.3 Exception to general rules

If the problematic axioms of an unsatisfiable class do not have a common error pattern, and we find no way to weaken the restrictions, the unsatisfiable class might be an exception to the general rules. This is because one common cause for inconsistencies in ontologies is that the new information to be incorporated is not compatible with the ontology. This problem also occurs in the ontology evolution process. Flouris and Plexousakis [46] present an abstract proposition that may resolve the problem in ontology evolution, based on the related field of belief change. Their approach is obviously applicable to the resolution of inconsistent ontologies as well.

Example 44 *Consider a typical penguin-bird example,*
1. $\mathsf{Bird} \sqsubseteq \mathsf{Fly}$
2. $\mathsf{Penguin} \sqsubseteq \neg\ \mathsf{Fly}$
3. $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$

Intuitively one would agree that one of the identity criteria for some entity to belong to the class bird is that it flies. However, there exists a concept, namely Penguin*, which most certainly has instances in the world, is a subtype of bird, and has as a property that it does not fly. Many treat* bird can fly *as a general phenomenon, but penguin as an exception to that. An approach to correcting these statements is to weaken the fact* bird can fly*. As Flouris and Plexousakis [46] suggest, we could replace axiom 2:* $\mathsf{Bird} \sqsubseteq \mathsf{Fly}$ *with* $\mathsf{Bird} \sqcap \neg\ \mathsf{Penguin} \sqsubseteq \mathsf{Fly}$*, which means birds but not penguins can fly. The resulting ontology is consistent and causes minimal loss of information (the Principle of Minimal Change).*

9.4.2.4 Summary

For the methods of weakening restrictions proposed by Plessers [122], no user study is conducted to show its practical usefulness for real world ontologies. The strategies derived from common errors proposed by Kalyanpur [91] were only evaluated with two ontologies. Also, no existing tool utilises exception to general rules to generate rewriting suggestions. One can simply combine and extend the above strategies to provide rewriting suggestions, and then conduct usability studies to show their practical usefulness.

However, there are often many options for correcting unsatisfiable concepts, but usually only a few of these options are intuitively acceptable [105]. It is impractical to list all possible rewriting options to the user without ranking the options or giving advice which ones are mostly likely to be correct. Therefore, similar to the ranking heuristics, we can make use of provenance information to provide suitable suggestions which reflect the circumstances. For example, suggestions derived from common errors mistakes should be given for non-expert users; suggestions derived from exceptions to general rules or weakening restrictions of concepts should be given to expert users.

9.5 Conclusion

The proposed methods for resolving inconsistencies in ontologies, namely, a fine-grained approach and a heuristic-based service, are a promising starting point for debugging ontologies. The usability results suggest that our methods are useful, especially for less experienced users of ontologies. The outcomes of this thesis should be valuable to both implementors of ontology management tools, and to ontology users. It is hoped that the fine-grained approach and heuristic-based service will provide a firm foundation for future research, leading to the development of interactive ontology debugging tools and methods which are useful to ontology modellers in a wide range of ontology applications.

Appendix A

Empirical Study: Phase One (Pilot Study)

A.1 Resolving Inconsistencies in Ontologies: Empirical Study 1a

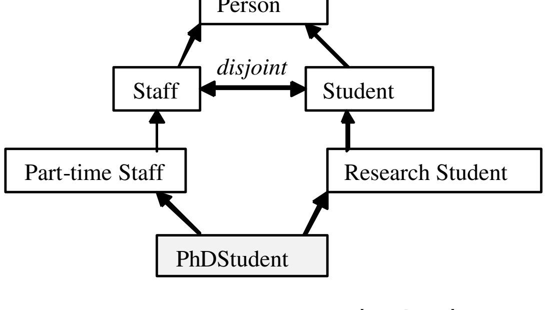
In this experiment, we want to analyse how you resolve inconsistencies in an ontology. We have to emphasise that we are interested in the strategies you use to solve problems, not in the particular solutions nor in assessing your ontological knowledge.

You are going to see a series of eight inconsistent ontologies, in which unsatisfiable concepts are highlighted. Each will be accompanied by a natural language explanation as well as graphical representation. While doing the task, you may be asked some questions by the investigator; for instance, why certain changes to a concept are suggested, and why other options are not considered. **Paper and pens will be provided and you can write anything you wish as you work through each task. Please however, note on the paper your solution to each task, and show as much intermediary working as possible.**

Sometimes a little warm-up helps, so let's try one practice problem first. Below we give the three forms for the ontological information: axioms, their corresponding natural language forms, and the corresponding graphical form.

Warming-up Task: The problematic axioms with natural language explanation are listed as below:

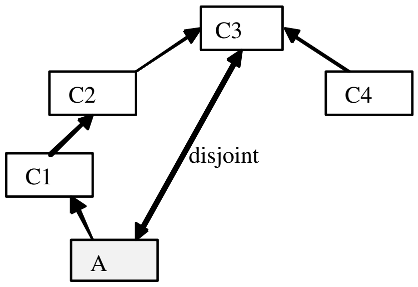
Staff $\sqsubseteq \neg$ Student	[All staff cannot be a student]
Part-time Staff $\sqsubseteq$ Staff	[Every part-time staff is a staff]
Research Student $\sqsubseteq$ Student	[Every research student is a student]
PhdStudent $\sqsubseteq$ Part-time Staff	[Every PhD student is a staff]
Research Student $\sqsubseteq$ Student	[Every PhD student is a student]



In the above ontology, PhDStudent is unsatisfiable, because it belongs to both Staff and Student, which are disjoint

You can make any change on the ontology, such as: changing the parent of a concept, or removing a concept, or removing a subsumption relationship.

The following task is similar to the above, but all the concepts included are abstract. We also have a warm-up task.



Warming-up Task:

Problematic axioms:

[**1**] $\mathsf{A} \sqsubseteq \mathsf{C1}$ [A is a subclass of C1],

[**2**] $\mathsf{C1} \sqsubseteq \mathsf{C2}$ [C1 is a subclass of C2],

[**3**] $\mathsf{C2} \sqsubseteq \mathsf{C3}$; [C2 is a subclass of C3],

[**4**] $\mathsf{A} \sqsubseteq \neg\, \mathsf{C3}$ [A is disjoint with C3]

Reason: A is unsatisfiable, as it belongs to class $\mathsf{C3}$ and its complement.

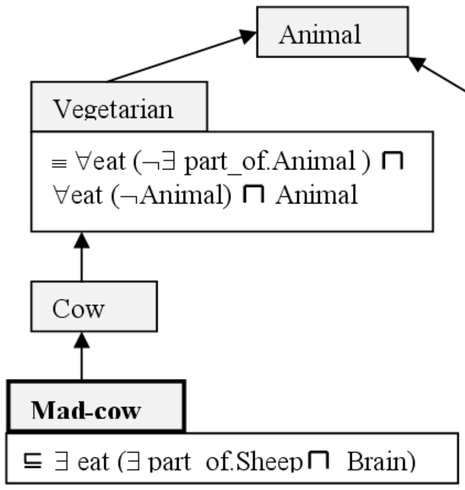
A.2 Main Task

The stimuli you are going to see are based on an animal ontology. Although the ontology is connected with description logics, this is not a test of your knowledge of description logics. This is a study of the strategies you use when you are faced with ontological problems. Your task is to make changes on the ontology in order to resolve the reported inconsistencies. Remember, we are more interested in the strategies you use, rather than your answers. Don’t worry if you feel that you contradict yourself when giving your answers. Again, during the task, you may take notes, and you may be asked by the investigator why a certain change is made, and why other options are not adopted.

After we finish each task, we will ask you a number of questions about the task.

Do you have any questions? Okay, let’s start with the following ontology.

Task 1


Animal
Vegetarian
≡ ∀eat (¬∃ part_of.Animal) ⊓ ∀eat (¬Animal) ⊓ Animal
Sheep
Cow
Mad-cow
⊑ ∃ eat (∃ part_of.Sheep ⊓ Brain)

Problematic axioms:

[**1**] Mad_cow $\sqsubseteq$ ∃eat.((∃part_of.Sheep) $\sqcap$ Brain) $\sqcap$ Cow

[A mad cow is a cow that eats the brains of sheep],

[**2**] Cow $\sqsubseteq$ Vegetarian [Cows are vegetarians],

[**3**] Vegetarian $\doteq$ (∀eat.(¬Animal)) $\sqcap$ Animal $\sqcap$ ∀eat.(¬∃ part_of.Animal)

[Vegetarians do not eat any part of animals],

[**4**] Sheep $\sqsubseteq$ Animals [Sheep are animals].

Reason: Mad_cow is unsatisfiable, because it is a vegetarian which does not eat any part of animals, but mad_cow itself eats sheep's brain.

Procedure

Subjects should attempt to work the individual tasks in the sequence in which they appear in this document. If a subject appears to be having difficulties, the experimenter should ask if the subject needs help and, if needed, some hints should be provided. Once the task is completed the experimenter should ask the subject to summarize their solution, and the experimenter should probe the solution a little (both when the task is correctly and incorrectly worked.) The subject is then asked to answer the following questions:

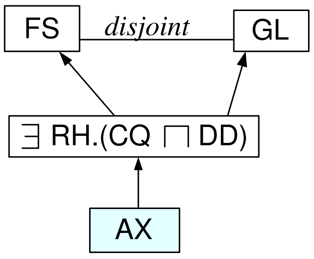
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 2



Problematic axioms:

[**1**] AX $\sqsubseteq \exists$ RH.(CQ $\sqcap$ DD) [At least one AX has RH relation with CQ and DD],

[**2**] CQ $\sqsubseteq$ FS [CQ is a subclass of FS],

[**3**] DD $\sqsubseteq$ GL [DD is a subclass of GL],

[**4**] GL $\sqsubseteq \neg$ FS [GL is disjoint with FS]

Reason: AX is unsatisfiable, because its RH relation filler is empty.
Questions:

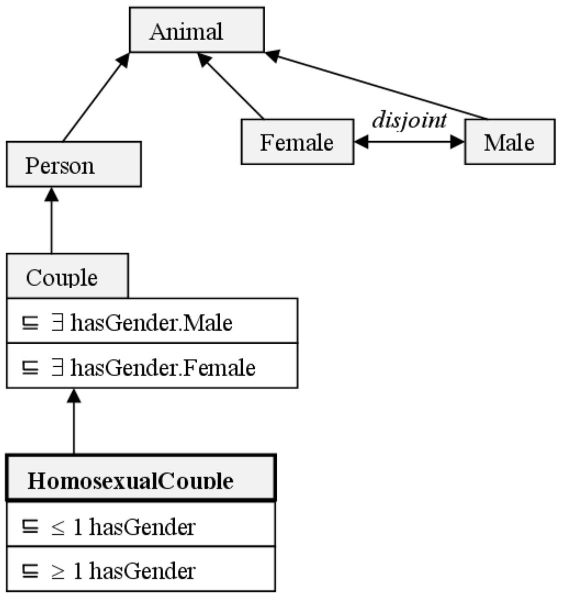
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 3



Problematic axioms:

[**1**] HomosexualCouple $\sqsubseteq$ ($\leq$ 1 hasGender) $\sqcap$($\geq$ 1 hasGender)

[A homosexual couple has exactly one gender],

[**2**] Couple $\sqsubseteq$ ($\exists$ hasGender.Male) $\sqcap$($\exists$ hasGender.Female)

[A couple has at least one male and one female],

[**3**] Male $\sqsubseteq$ $\neg$ Female [A male cannot be a female, and vice versa],

[**4**] HomosexualCouple $\sqsubseteq$ Couple [A homosexual couple is a type of couple].

Reason: HomosexualCouple is unsatisfiable, as it only has exactly one gender, but it also belongs to the class Couple, which has at least a male and a female, where male and female are disjoint.

Questions:

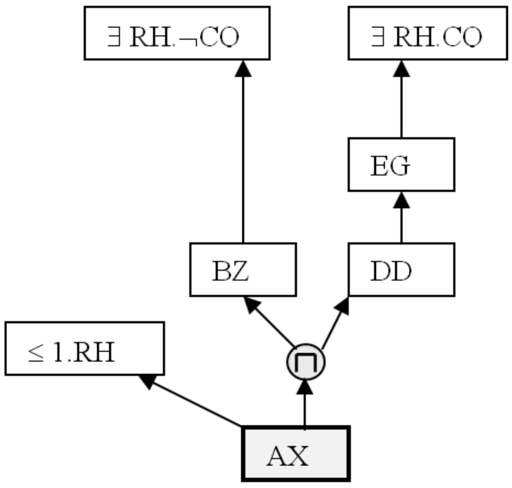
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 4



Problematic axioms:

[**1**] AX $\sqsubseteq \leq 1.$ RH [At most one AX has RH relation],

[**2**] AX $\sqsubseteq$ BZ $\sqcap$ DD [AX is the intersection of BZ and DD],

[**3**] DD $\sqsubseteq$ EG [DD is a subclass of EG],

[**4**] EG $\sqsubseteq \exists$ RH.CQ [At least one EG has RH relation with CQ],

[**5**] BZ $\sqsubseteq \exists$ RH.$\neg$ CQ [All EG has no RH relation with CQ]

Reason: AX is unsatisfiable, as it has at most one RH relation, and has at least two RH relations simultaneously.

Questions:

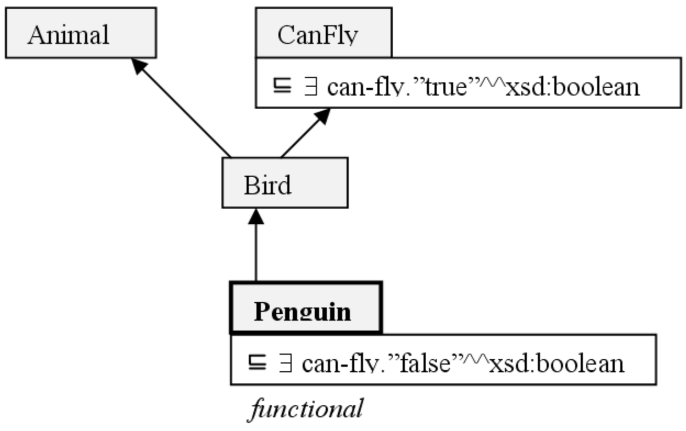
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 5


Animal
CanFly
⊑ ∃ can-fly."true"^^xsd:boolean
Bird
Penguin
⊑ ∃ can-fly."false"^^xsd:boolean
functional

Problematic axioms:

[1] CanFly ⊑ (∃ can-fly."true"^^xsd:boolean) [The individuals of CanFly can fly],

[2] Bird ⊑ CanFly [Bird belongs to CanFly],

[3] Penguin ⊑ Bird [Penguin belongs to Bird],

[4] Penguin ⊑ (∃ can-fly."false"^^xsd:boolean) [A penguin cannot fly],

[5] functional(can-fly) [can-fly is a functional property].

Reason: Penguin is unsatisfiable , as it is a type of birds and cannot fly, but birds can fly.

Questions:

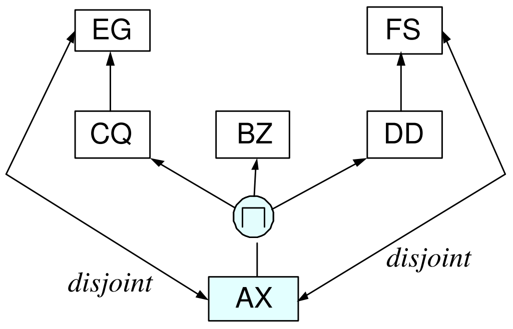
[1] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[2] If you were to have only a single form which would it be?

[3] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[4] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 6



Problematic axioms:

[**1**] AX ⊑ BZ ⊓ CQ ⊓ DD [AX is the intersection of BZ, CQ and DD],

[**2**] CQ ⊑ EG [CQ is a subclass of EG],

[**3**] EG ⊑ ¬ AX; [EG is disjoint with AX],

[**4**] FS ⊑ ¬ AX [FS is disjoint with AX],

[**5**] DD ⊑ FS; [DD is a subclass of FS]

Reason: AX is unsatisfiable, as it is disjoint with its superconcepts EG and FS.

Questions:

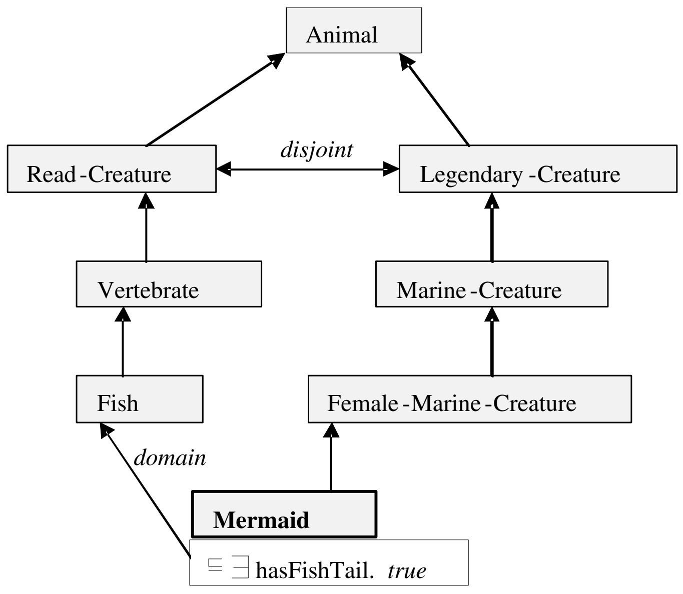
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 7



Problematic axioms:

[**1**] Mermaid ⊑ ∃ hasFishTail.{“true”^^xsd:boolean} [A mermaid has a fish tail],

[**2**] Mermaid ⊑ Female-Marine-Creature [A mermaid is a female marine creature],

[**3**] ∃ hasFishTail.⊤ ⊑ Fish; [The domain of hasFishTail is Fish],

[**4**] Real-Creature ⊑ ¬ Legendary-Creature [A real creature cannot be a a legendary creature],

[**5**] Fish ⊑ Vertebrate [A fish is vertebrate],

[**6**] Vertebrate ⊑ Real-Creature [A vertebrate is a real creature],

[**7**] Marine-Creature ⊑ Legendary-Creature [A marine creature is a legendary creature].

Reason: Mermaid belongs to class Real-Creature and its complement Legendary-Creature.

Questions:

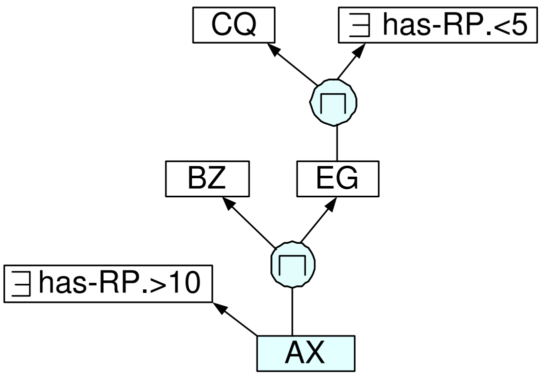
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 8



Problematic axioms:

[**1**] AX $\sqsubseteq \exists$ has-RP.$>$ 10

[AX has RP relation whose value is great than 10],

[**2**] AX $\sqsubseteq$ BZ $\sqcap$ EG [AX is the intersection of BZ and EG],

[**3**] EG $\sqsubseteq$ CQ $\sqcap \exists$ RP.$<$5

[EG is a type of CQ, and has RP relation whose value is less than 5]

Reason: AX is unsatisfiable, as its has-RP relation has values greater than 10 and less than 5 simultaneously.

Questions:

[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

A.3 Resolving Inconsistencies in Ontologies: Empirical Study 1b

In this experiment, we want to analyse how you resolve inconsistencies in an ontology. We have to emphasise that we are interested in the strategies you use to solve problems, not in the particular solutions nor in assessing your ontological knowledge.

You are going to see a series of eight inconsistent ontologies, in which unsatisfiable concepts are highlighted. Each will be accompanied by a natural language explanation as well as graphical representation. While doing the task, you may be asked some questions by the investigator; for instance, why certain changes to a concept are suggested, and why other options are not considered. **Paper and pens will be provided and you can write anything you wish as you work through each task. Please however, note on the paper your solution to each task, and show as much intermediary working as possible.**

Sometimes a little warm-up helps, so let's try one practice problem first. Below we give the three forms for the ontological information: axioms, their corresponding natural language forms, and the corresponding graphical form.

Warming-up Task: The problematic axioms with natural language explanation are listed as below:

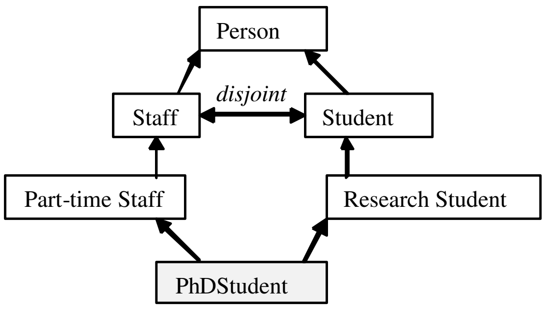
[**1**] Staff $\sqsubseteq \neg$ Student [All staff cannot be a student]

[**2**] Part-time Staff $\sqsubseteq$ Staff [Every part-time staff is a staff]

[**3**] Research Student $\sqsubseteq$ Student [Every research student is a student

[**4**] PhdStudent $\sqsubseteq$ Part-time Staff [Every PhD student is a staff]

[**5**] Research Student $\sqsubseteq$ Student [Every PhD student is a student]



In the above ontology, PhDStudent is unsatisfiable, because it belongs to both Staff and Student, which are disjoint

You can make any change on the ontology, such as: changing the parent of a concept, or removing a concept, or removing a subsumption relationship.

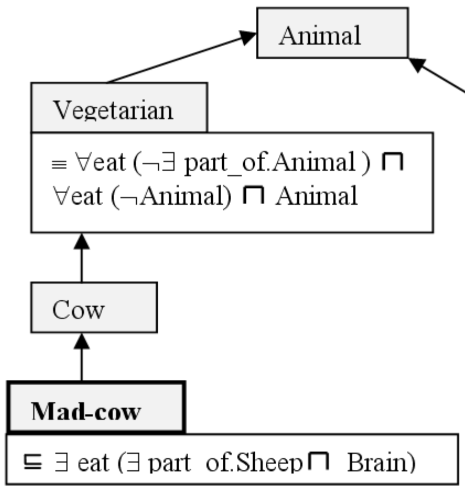
A.4 Main Task

The stimuli you are going to see are based on an animal ontology. Although the ontology is connected with description logics, this is not a test of your knowledge of description logics. This is a study of the strategies you use when you are faced with ontological problems. Your task is to make changes on the ontology in order to resolve the reported inconsistencies. Remember, we are more interested in the strategies you use, rather than your answers. Don't worry if you feel that you contradict yourself when giving your answers. Again, during the task, you may take notes, and you may be asked by the investigator why a certain change is made, and why other options are not adopted.

After we finish each task, we will ask you a number of questions about the task.

Do you have any questions? Okay, let's start with the following ontology.

Task 1


Animal
Vegetarian
≡ ∀eat (¬∃ part_of.Animal) ⊓ ∀eat (¬Animal) ⊓ Animal
Sheep
Cow
Mad-cow
⊑ ∃ eat (∃ part_of.Sheep ⊓ Brain)

Problematic axioms:

[1] $\text{Mad\_cow} \sqsubseteq \exists\text{eat}.((\exists\text{part\_of.Sheep}) \sqcap \text{Brain}) \sqcap \text{Cow}$

[A mad cow is a cow that eats the brains of sheep],

[2] $\text{Cow} \sqsubseteq \text{Vegetarian}$ [Cows are vegetarians],

[3] $\text{Vegetarian} \doteq \forall\text{eat}.(\neg\text{Animal}) \sqcap \text{Animal} \sqcap \forall\text{eat}.(\neg\exists\ \text{part\_of.Animal})$

[Vegetarians do not eat any part of animals],

[4] $\text{Sheep} \sqsubseteq \text{Animal}$ [Sheep are animals].

Reason: Mad_cow is unsatisfiable, because it is a vegetarian which does not eat any part of animals, but mad_cow itself eats sheep's brain.

Procedure

Subjects should attempt to work the individual tasks in the sequence in which they appear in this document. If a subject appears to be having difficulties, the experimenter should ask if the subject needs help and, if needed, some hints should be provided. Once the task is completed the experimenter should ask the subject to summarize their solution, and the experimenter should probe the solution a little (both when the task is correctly and incorrectly worked.) The subject is then asked to answer the following questions:

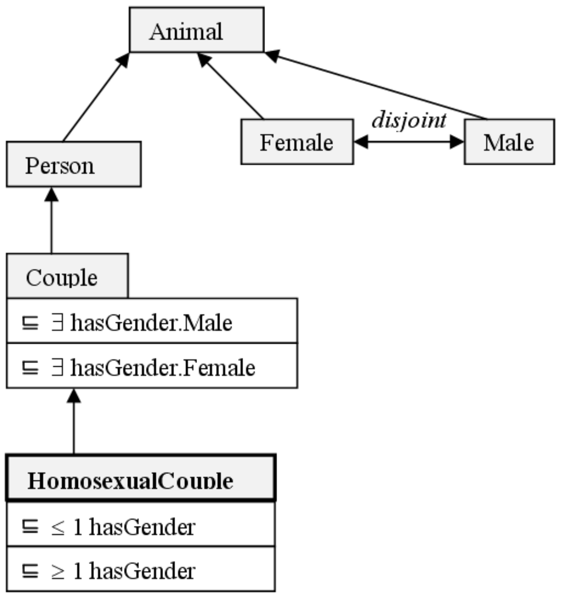
[1] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[2] If you were to have only a single form which would it be?

[3] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[4] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 2



Problematic axioms:

[**1**] $\mathsf{HomosexualCouple} \sqsubseteq (\leq 1\ \mathsf{hasGender}) \sqcap (\geq 1\ \mathsf{hasGender})$

[A homosexual couple has exactly one gender],

[**2**] $\mathsf{Couple} \sqsubseteq (\exists\ \mathsf{hasGender.Male}) \sqcap (\exists\ \mathsf{hasGender.Female})$

[A couple has at least one male and one female],

[**3**] $\mathsf{Male} \sqsubseteq \neg\ \mathsf{Female}$ [A male cannot be a female, and vice versa],

[**4**] $\mathsf{HomosexualCouple} \sqsubseteq \mathsf{Couple}$ [A homosexual couple is a type of couple].

Reason: HomosexualCouple is unsatisfiable, as it only has exactly one gender, but it also belongs to the class Couple, which has at least a male and a female, where male and female are disjoint.

Questions:

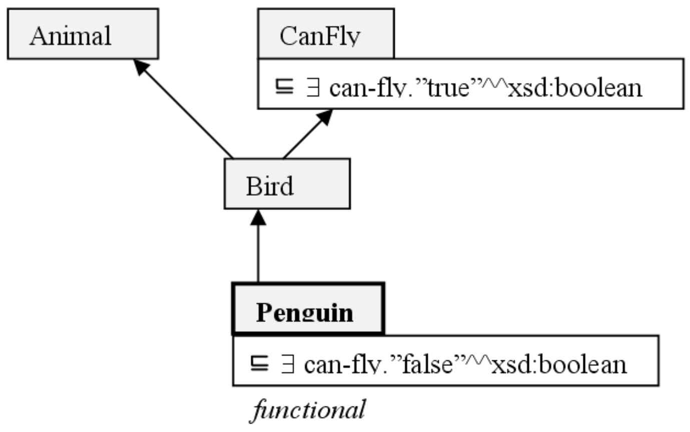
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 3


Animal
CanFly
⊑ ∃ can-fly."true"^^xsd:boolean
Bird
Penguin
⊑ ∃ can-fly."false"^^xsd:boolean
functional

Problematic axioms:

[**1**] CanFly ⊑ (∃ can-fly."true"^^xsd:boolean) [The individuals of CanFly can fly],

[**2**] Bird ⊑ CanFly [Bird belongs to CanFly],

[**3**] Penguin ⊑ Bird [Penguin belongs to Bird],

[**4**] Penguin ⊑ (∃ can-fly."false"^^xsd:boolean) [A penguin cannot fly],

[**5**] functional(can-fly) [can-fly is a functional property].

Reason: Penguin is unsatisfiable , as it is a type of birds and cannot fly, but birds can fly.
Questions:

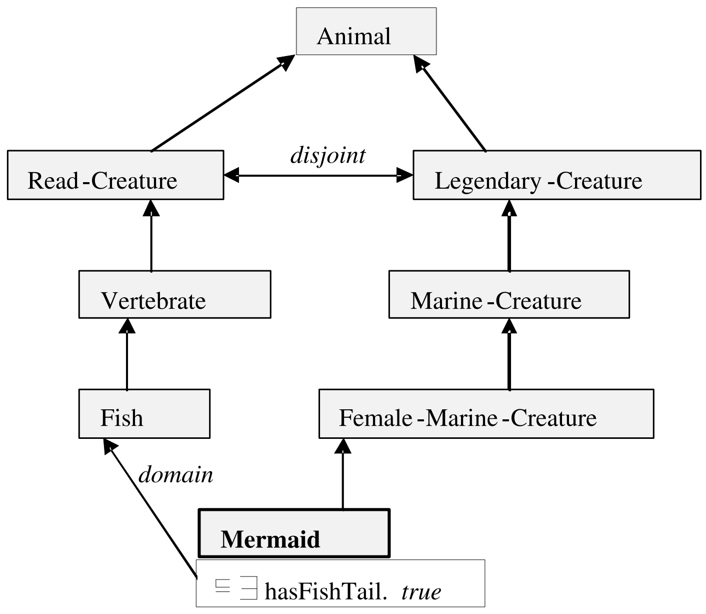
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 4



Problematic axioms:

[1] Mermaid $\sqsubseteq \exists$ hasFishTail.{“true”^^xsd:boolean} [A mermaid has a fish tail],

[2] Mermaid $\sqsubseteq$ Female-Marine-Creature [A mermaid is a female marine creature],

[3] $\exists$ hasFishTail.$\top \sqsubseteq$ Fish; [The domain of hasFishTail is Fish],

[4] Real-Creature $\sqsubseteq \neg$ Legendary-Creature [A real creature cannot be a a legendary creature],

[5] Fish $\sqsubseteq$ Vertebrate [A fish is vertebrate],

[6] Vertebrate $\sqsubseteq$ Real-Creature [A vertebrate is a real creature],

[7] Marine-Creature $\sqsubseteq$ Legendary-Creature [A marine creature is a legendary creature].

Reason: Mermaid belongs to class Real-Creature and its complement Legendary-Creature.

Questions:

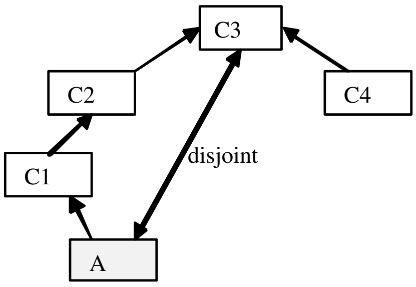
[1] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[2] If you were to have only a single form which would it be?

[3] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[4] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

The following task is similar to the above, but all the concepts included are abstract. We also have a warm-up task.



Warming-up Task:

Problematic axioms:

[**1**] $\mathsf{A} \sqsubseteq \mathsf{C1}$ [A is a subclass of C1],

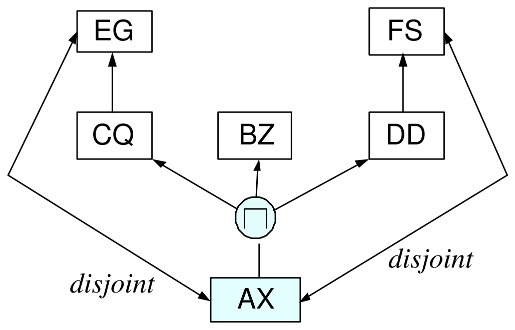
[**2**] $\mathsf{C1} \sqsubseteq \mathsf{C2}$ [C1 is a subclass of C2],

[**3**] $\mathsf{C2} \sqsubseteq \mathsf{C3}$; [C2 is a subclass of C3],

[**4**] $\mathsf{A} \sqsubseteq \neg\, \mathsf{C3}$ [A is disjoint with C3]

Reason: A is unsatisfiable, as it belongs to class $\mathsf{C3}$ and its complement.

Task 5



Problematic axioms:

[**1**] $\mathsf{AX} \sqsubseteq \mathsf{BZ} \sqcap \mathsf{CQ} \sqcap \mathsf{DD}$ [AX is the intersection of BZ, CQ and DD],

[**2**] $\mathsf{CQ} \sqsubseteq \mathsf{EG}$ [CQ is a subclass of EG],

[**3**] $\mathsf{EG} \sqsubseteq \neg\, \mathsf{AX}$; [EG is disjoint with AX],

[**4**] $\mathsf{FS} \sqsubseteq \neg\, \mathsf{AX}$ [FS is disjoint with AX],

[**5**] $\mathsf{DD} \sqsubseteq \mathsf{FS}$; [DD is a subclass of FS]

Reason: AX is unsatisfiable, as it is disjoint with its superconcepts EG and FS.
Questions:

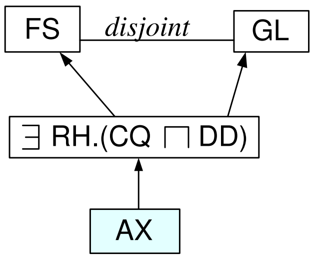
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 6



Problematic axioms:

[**1**] $\mathsf{AX} \sqsubseteq \exists\, \mathsf{RH}.(\mathsf{CQ} \sqcap \mathsf{DD})$ [At least one AX has RH relation with CQ and DD],

[**2**] $\mathsf{CQ} \sqsubseteq \mathsf{FS}$ [CQ is a subclass of FS],

[**3**] $\mathsf{DD} \sqsubseteq \mathsf{GL}$ [DD is a subclass of GL],

[**4**] $\mathsf{GL} \sqsubseteq \neg\, \mathsf{FS}$ [GL is disjoint with FS]

Reason: AX is unsatisfiable, because its RH relation filler is empty.

Questions:

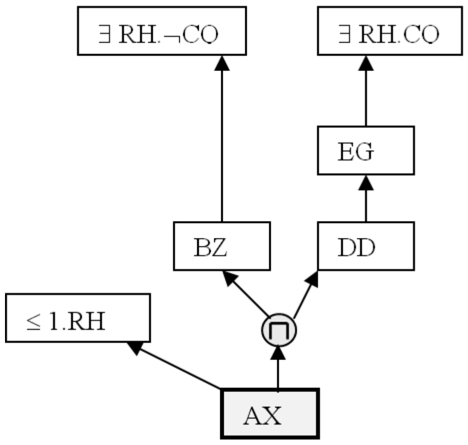
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 7



Problematic axioms:

[**1**] $\mathsf{AX} \sqsubseteq \leq 1.\ \mathsf{RH}$ [At most one AX has RH relation],

[**2**] $\mathsf{AX} \sqsubseteq \mathsf{BZ} \sqcap \mathsf{DD}$ [AX is the intersection of BZ and DD],

[**3**] $\mathsf{DD} \sqsubseteq \mathsf{EG}$ [DD is a subclass of EG],

[**4**] $\mathsf{EG} \sqsubseteq \exists\ \mathsf{RH.CQ}$ [At least one EG has RH relation with CQ],

[**5**] $\mathsf{BZ} \sqsubseteq \exists\ \mathsf{RH}.\neg\ \mathsf{CQ}$ [All EG has no RH relation with CQ]

Reason: AX is unsatisfiable, as it has at most one RH relation, and has at least two RH relations simultaneously.
Questions:

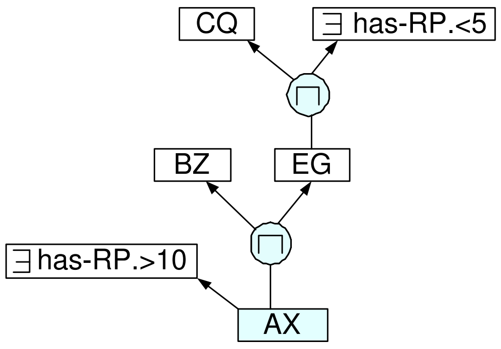
[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Task 8



Problematic axioms:

[**1**] $\mathsf{AX} \sqsubseteq \exists\ \mathsf{has\text{-}RP}. > 10$

[AX has RP relation whose value is great than 10],

[**2**] $\mathsf{AX} \sqsubseteq \mathsf{BZ} \sqcap \mathsf{EG}$ [AX is the intersection of BZ and EG],

[**3**] $\mathsf{EG} \sqsubseteq \mathsf{CQ} \sqcap \exists\ \mathsf{RP}. < 5$

[EG is a type of CQ, and has RP relation whose value is less than 5]

Reason: AX is unsatisfiable, as its has-RP relation has values greater than 10 and less than 5 simultaneously.

Questions:

[**1**] Classify each presentational form (axioms, natural language and graphical) on a 5 point scale where 5 is vital and 1 corresponds to no help.

[**2**] If you were to have only a single form which would it be?

[**3**] Do you think that certain pairs are very helpful/ supportive of each other? Which?

[**4**] Do you have any other comments about this particular task? i.e. how it was presented or the actual task?

Appendix B

Empirical Study: Phase Two

B.1 Resolving Inconsistencies in Ontologies

In this experiment, we want to analyse how you resolve inconsistencies in an ontology. We have to emphasise that we are interested in the strategies you use to solve problems, not in the particular solutions nor in assessing your ontological knowledge.

You are going to see a series of five (four concrete and one abstract) inconsistent ontologies, in which unsatisfiable concepts are highlighted. Each will be accompanied by a natural language explanation as well as graphical representation. While doing the task, you may be asked some questions by the investigator; for instance, why certain changes to a concept are suggested, and why other options are not considered. **Paper and pens will be provided and you can write anything you wish as you work through each task. Please however, note on the paper your solution to each task, and show as much intermediary working as possible.**

Sometimes a little warm-up helps, so let's try one practice problem first. Below we give the three forms for the ontological information: axioms, their corresponding natural language forms, and the corresponding graphical form.

Warming-up Task: The problematic axioms with natural language explanation are listed as below:

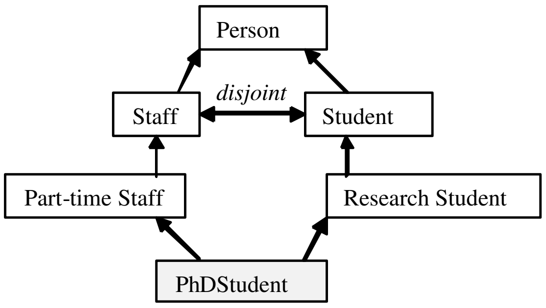
[**1**] Staff $\sqsubseteq \neg$ Student [All staff cannot be a student]

[**2**] Part-time Staff $\sqsubseteq$ Staff [Every part-time staff is a staff]

[**3**] Research Student $\sqsubseteq$ Student [Every research student is a student

[**4**] PhdStudent $\sqsubseteq$ Part-time Staff [Every PhD student is a staff]

[**5**] Research Student $\sqsubseteq$ Student [Every PhD student is a student]



In the above ontology, PhDStudent is unsatisfiable, because it belongs to both Staff and Student, which are disjoint

You can make any change on the ontology, such as: changing the parent of a concept, or removing a concept, or removing a subsumption relationship.

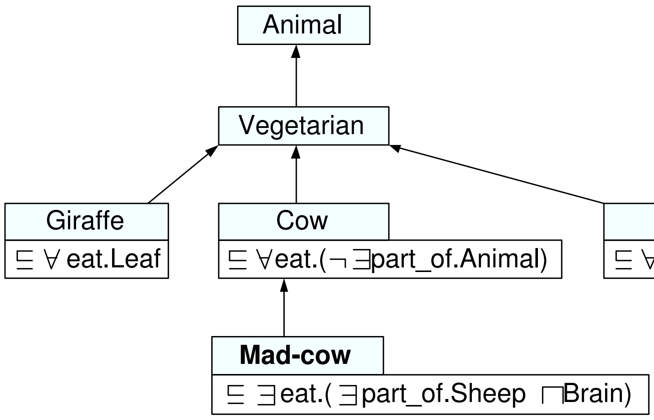
B.2 Main Task

The stimuli you are going to see are based on an animal ontology. Although the ontology is connected with description logics, this is not a test of your knowledge of description logics. This is a study of the strategies you use when you are faced with ontological problems. Your task is to make changes on the ontology in order to resolve the reported inconsistencies. Remember, we are more interested in the strategies you use, rather than your answers. Don't worry if you feel that you contradict yourself when giving your answers. Again, during the task, you may take notes, and you may be asked by the investigator why a certain change is made, and why other options are not adopted.

After we finish each task, we will ask you a number of questions about the task.

Do you have any questions? Okay, let's start with the following ontology.

Task 1



Problematic axioms:

[**1**] $\mathsf{Mad\_cow} \sqsubseteq \exists \mathsf{eat.}((\exists\ \mathsf{part\_of.Sheep}) \sqcap \mathsf{Brain})$

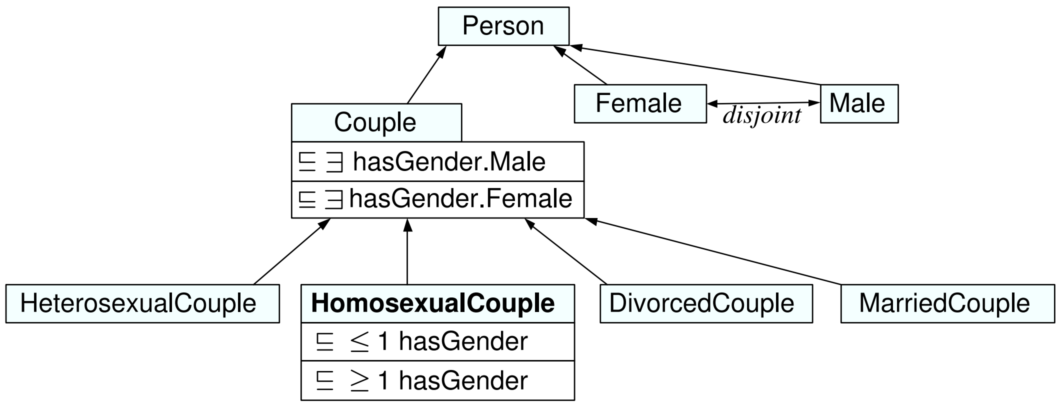
[A mad cow eats the brains of sheep],

[**2**] $\mathsf{Mad\_Cow} \sqsubseteq \mathsf{Cow}$ [Mad cows are cows],

[**3**] $\mathsf{Cow} \sqsubseteq \forall \mathsf{eat.}(\neg\exists\ \mathsf{part\_of.Animal})$ [Cows do not eat any part of animals],

[**4**] $\mathsf{Sheep} \sqsubseteq \mathsf{Vegetarian}$ [Sheep are vegetarians]

Task 2



Problematic axioms:

[**1**] HomosexualCouple $\sqsubseteq$ ($\leq$ 1 hasGender) $\sqcap$($\geq$ 1 hasGender)

[A homosexual couple has exactly one gender],

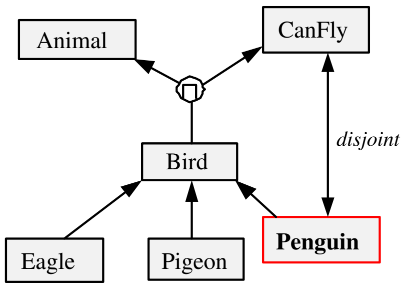
[**2**] Couple $\sqsubseteq$ ($\exists$ hasGender.Male) $\sqcap$($\exists$ hasGender.Female)

[A couple has at least one male and one female],

[**3**] Male $\sqsubseteq$ $\neg$ Female [A male cannot be a female, and vice versa],

[**4**] HomosexualCouple $\sqsubseteq$ Couple [A homosexual couple is a type of couple].

Task 3



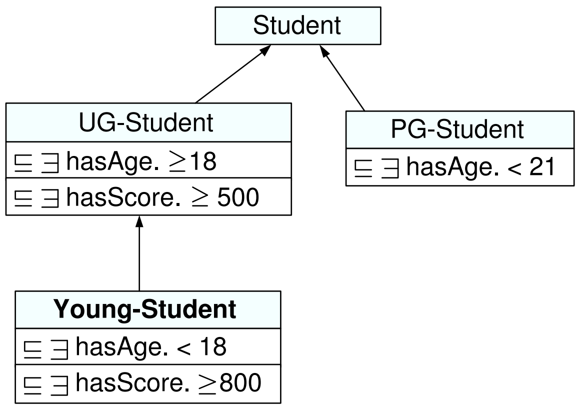
Problematic axioms:

[**1**] Bird $\sqsubseteq$ Animal $\sqcap$ CanFly [Birds are animals which can fly],

[**2**] Penguin $\sqsubseteq$ Bird [Penguins are birds],

[**3**] Penguin $\sqsubseteq$ $\neg$ CanFly [Penguins cannot fly]

Task 4



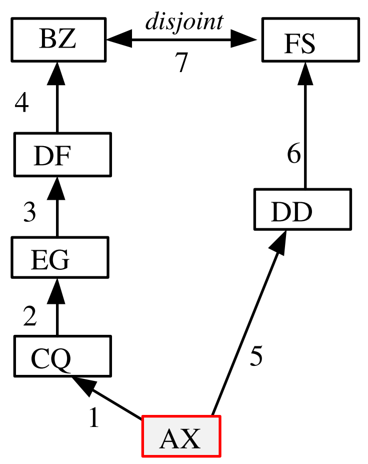
Problematic axioms:

[**1**] UG-Student $\sqsubseteq \exists$ hasAge.$\geq$18 [The minimum required age of a UG student is 18],

[**2**] Young-Student $\sqsubseteq$ UG-Student [A young student is a UG student],

[**3**] Young-Student $\sqsubseteq \exists$hasAge.$<$ 18 [A young student is less than 18]

Task 5



Problematic Axioms:

[**1**] AX $\sqsubseteq$CQ

[**2**] CQ $\sqsubseteq$ EG

[**3**] EG $\sqsubseteq$ DF

[**4**] DF $\sqsubseteq$ BZ

[**5**] AX $\sqsubseteq$ DD

[**6**] DD $\sqsubseteq$ FS

[**7**] FS $\sqsubseteq$ $\neg$ BZ

Appendix C

Empirical Study: Phase Three

C.1 Resolving Inconsistencies in Ontologies

In this experiment, we want to analyse how you resolve inconsistencies in an ontology. We have to emphasise that we are interested in the strategies you use to solve problems, not in the particular solutions nor in assessing your ontological knowledge.

You are going to see a series of five abstract inconsistent ontologies, in which unsatisfiable concepts are highlighted. Each will be accompanied by a natural language explanation as well as graphical representation. While doing the task, you may be asked some questions by the investigator; for instance, why certain changes to a concept are suggested, and why other options are not considered. **Paper and pens will be provided and you can write anything you wish as you work through each task. Please however, note on the paper your solution to each task, and show as much intermediary working as possible.**

Sometimes a little warm-up helps, so let's try one practice problem first. Below we give the three forms for the ontological information: axioms, their corresponding natural language forms, and the corresponding graphical form.

Warming-up Task: The problematic axioms with natural language explanation are listed as below:

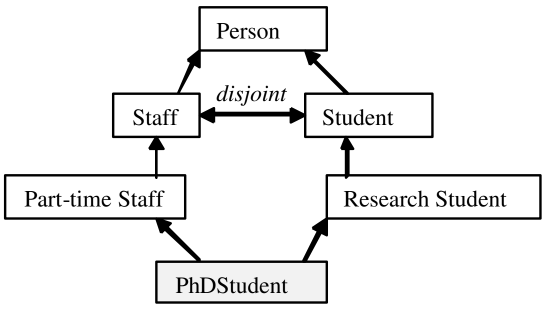
[**1**] Staff $\sqsubseteq \neg$ Student [All staff cannot be a student]

[**2**] Part-time Staff $\sqsubseteq$ Staff [Every part-time staff is a staff]

[**3**] Research Student $\sqsubseteq$ Student [Every research student is a student

[**4**] PhdStudent $\sqsubseteq$ Part-time Staff [Every PhD student is a staff]

[**5**] Research Student $\sqsubseteq$ Student [Every PhD student is a student]



In the above ontology, PhDStudent is unsatisfiable, because it belongs to both Staff and Student, which are disjoint

You can make any change on the ontology, such as: changing the parent of a concept, or removing a concept, or removing a subsumption relationship.

C.2 Main Task

The stimuli you are going to see are based on an animal ontology. Although the ontology is connected with description logics, this is not a test of your knowledge of description logics. This is a study of the strategies you use when you are faced with ontological problems. Your task is to make changes on the ontology in order to resolve the reported inconsistencies. Remember, we are more interested in the strategies you use, rather than your answers. Don't worry if you feel that you contradict yourself when giving your answers. Again, during the task, you may take notes, and you may be asked by the investigator why a certain change is made, and why other options are not adopted.

After we finish each task, we will ask you a number of questions about the task.

Do you have any questions? Okay, let's start with the following ontology.

Task 1

The following three axioms cause concept A to be unsatisfiable because it is a subclass of C and $\neg C$.

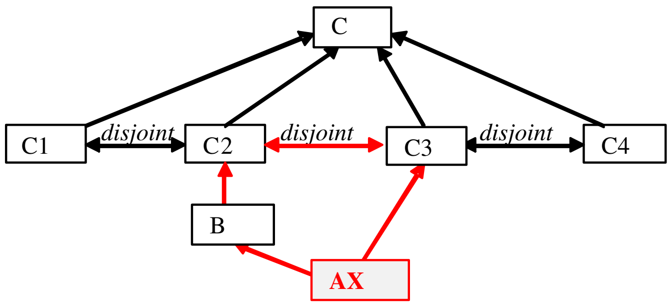
[1] $\mathsf{B} \sqsubseteq \neg\, \mathsf{C}$

[2] $\mathsf{A} \sqsubseteq \mathsf{C} \sqcap \mathsf{D} \sqcap \exists\, \mathsf{R}.(\mathsf{C} \sqcup \mathsf{D}) \sqcap \mathsf{F} \sqcap \mathsf{G}$

[3] $\mathsf{D} \sqsubseteq \mathsf{B} \sqcap \exists\, \mathsf{R}.(\mathsf{E} \sqcup \mathsf{F})$

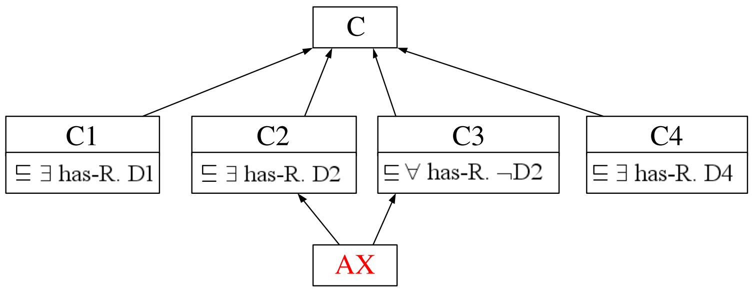
Which axiom will you remove to resolve the unsatisfiability of A?

Task 2



AX is unsatisfiable because the instances of AX belong to class C2 and C3 simultaneously, but C2 and C3 are disjoint with each other.
Problematic Axioms:
1. AX $\sqsubseteq$ B
2. AX $\sqsubseteq$ C3
3. B $\sqsubseteq$ C2
4. C3 $\sqsubseteq \neg$ C2

Task 3



AX is unsatisfiable because at least an instance of AX participates in has-R with D2, but all of its instances do not participate in has-R with D2.
Problematic axioms:

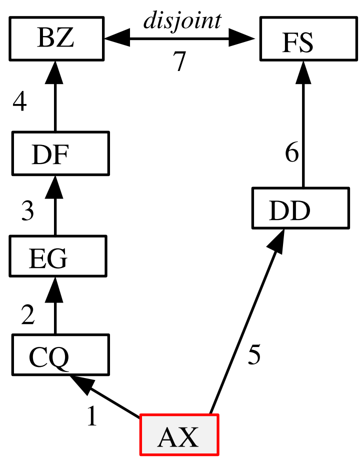
[1] AX ⊑ C2,

[2] AX ⊑ C3,

[3] C2 ⊑ ∃ has-R.D2

[4] C3 ⊑ ∀ has-R.¬D2

Task 4



AX is unsatisfiable because it is a subclass of BZ and ¬ BZ.
Problematic Axioms:

[1] AX ⊑CQ

[2] CQ ⊑ EG

[3] EG ⊑ DF

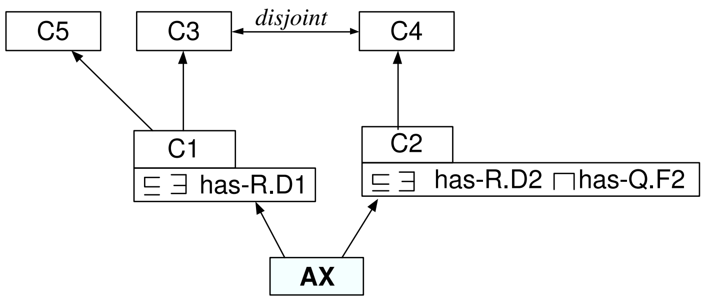
[4] DF ⊑ BZ

[5] AX ⊑ DD

[6] DD ⊑ FS

[7] FS ⊑ ¬ BZ

Task 5



AX is unsatisfiable because it is a subclass of both C3 and C4, which are disjoint with each other.
Problematic Axioms:

[**1**] AX $\sqsubseteq$ C1

[**2**] AX $\sqsubseteq$ C2

[**3**] C1 $\sqsubseteq$ C3

[**4**] C2 $\sqsubseteq$ C4

[**5**] C3 $\sqsubseteq \neg$ C4

Appendix D

Evaluation on Fine-grained Approach – Practical Worksheet

D.1 Practical 10: Resolving unsatisfiable ontologies using RepairTab

	Group A
Name	
Date	

D.1.1 Overview

The RepairOntologyTab plugin is a reasoner which supports you to resolve the unsatisfiable concepts in OWL ontologies.

In this practical, you can use a sample ontology to become familiar with the functionalities of the plugin. You will then be given five ontologies; using different functionalities of the plugin you will resolve the problems in the ontologies.

In the exercise, you are assigned to different groups; each group will have different functionalities in the plugin available to them. You are asked to resolve the given ontologies and answer the questions in the questionnaire. The following is an example ontology to illustrate how to use the plugin.

OWL ontology files:
1. Download the OWL ontology files from http://www.csd.abdn.ac.uk/ slam/owl/Onto.zip
2. Save and unzip the files to Local Disk (S:) in your computer.

Instructions of using the plugin
You can go to http://www.csd.abdn.ac.uk/ slam/practical/practical10.html to go through the steps of importing an OWL ontology into Protégé.

The following steps describe how to use the RepairTab plugin.

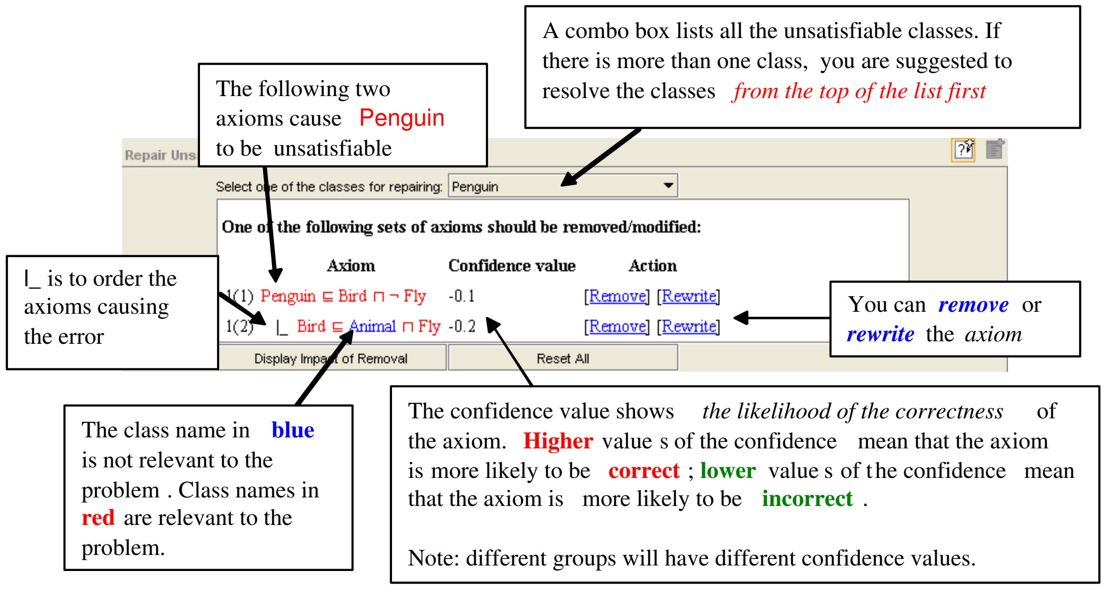
[**1**] Load the OWL ontology file Bird.owl into Protégé.

[**2**] Go to the `Project` in the menu bar, choose `Configure` ...

[**3**] Select the checkbox `RepairTab`, then click OK.

[**4**] Go to the `RepairTab`, click the icon $?^{+}$ on the right hand side of the screen, and wait for a while.

[**5**] The following screen will be displayed:


A combo box lists all the unsatisfiable classes. If there is more than one class, you are suggested to resolve the classes *from the top of the list first*

The following two axioms cause Penguin to be unsatisfiable

|_ is to order the axioms causing the error

You can ***remove*** or ***rewrite*** the *axiom*

The class name in **blue** is not relevant to the problem. Class names in **red** are relevant to the problem.

The confidence value shows *the likelihood of the correctness* of the axiom. **Higher** value s of the confidence mean that the axiom is more likely to be **correct** ; **lower** value s of the confidence mean that the axiom is more likely to be **incorrect** .

Note: different groups will have different confidence values.

Repair Uns

Select one of the classes for repairing: Penguin

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	Penguin ⊑ Bird ⊓ ¬ Fly	-0.1	[Remove] [Rewrite]
1(2)	|_ Bird ⊑ Animal ⊓ Fly	-0.2	[Remove] [Rewrite]

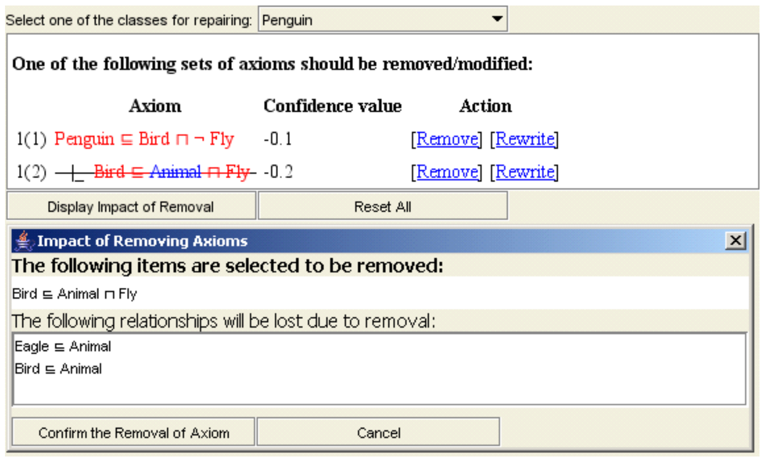
[**6**] The above figure shows that the axiom Bird ⊑ Animal ⊓ Fly has a lower confidence value, that means, the system suggests that axiom is likely to be incorrect. You can follow or ignore the suggestions given by the system to resolve the problem.

[**7**] Two types of actions are available: **[Remove], [Rewrite]**

[**8**] Click on the link **[Remove]**; the axiom will be struck out.

[**9**] Note that there is no UNDO available, if you have struck out an axiom, and you change your mind, you have to click `Reset All`.

[**10**] Click on `Display Impact of Removal`. The following window displaying the `Impact of Removing Axiom` will pop up. This shows what relationships will be lost if the axiom is removed. In our example, it shows that if Bird ⊑ Animal ⊓ Fly is removed, then the relationships Bird ⊑ Animal and Eagle ⊑ Animal will be lost.


Select one of the classes for repairing: Penguin

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	Penguin ⊑ Bird ⊓ ¬ Fly	-0.1	[Remove] [Rewrite]
1(2)	~~|_ Bird ⊑ Animal ⊓ Fly~~	-0.2	[Remove] [Rewrite]

Display Impact of Removal
Reset All
Impact of Removing Axioms

The following items are selected to be removed:

Bird ⊑ Animal ⊓ Fly

The following relationships will be lost due to removal:

Eagle ⊑ Animal
Bird ⊑ Animal

Confirm the Removal of Axiom
Cancel

[11] You can click on the button `Confirm the Removal of Axiom` to execute the removal of this axiom from the ontology, or you can click on the `Cancel` button to cancel the removal.

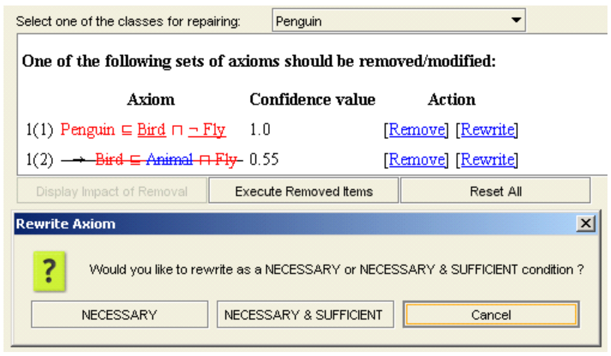
[12] Note that there is no UNDO available, if you have executed the removal of axioms, the plugin cannot undo the removal. You can re-load the ontology into Protégé and restart again.

[13] Let us click on the `Cancel` button to cancel the removal, and try to rewrite axioms.

[14] Click on **[Rewrite]**, you can type your own axiom to replace the old one.

[15] In the following figure, it shows that Bird ⊑ Animal ⊓ Fly has lower confidence value. You can take the value as a reference for rewriting if you prefer.

[16] You have to decide to rewrite the axiom as a NECESSARY (Bird ⊑ Animal ⊓ Fly) or NECESSARY & SUFFICIENT (Bird ≐ Animal ⊓ Fly) condition.



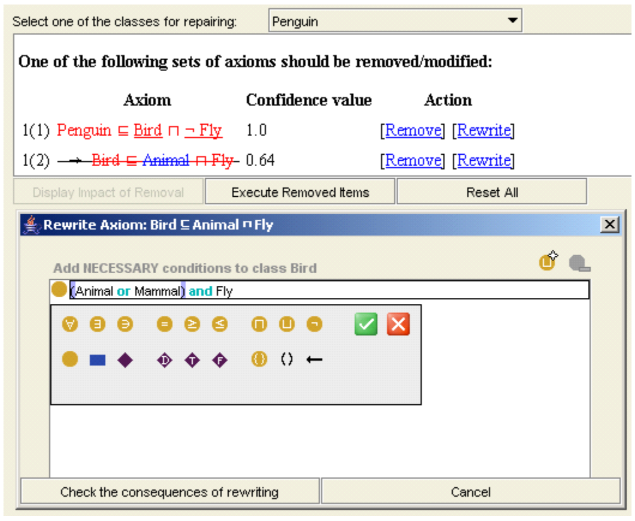
Select one of the classes for repairing: Penguin

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	Penguin ⊑ Bird ⊓ ¬ Fly	1.0	[Remove] [Rewrite]
1(2)	→ ~~Bird ⊑ Animal ⊓ Fly~~	0.55	[Remove] [Rewrite]

Would you like to rewrite as a NECESSARY or NECESSARY & SUFFICIENT condition ?

[17] Once you have chosen the condition, the following screen will pop up:



Select one of the classes for repairing: Penguin

One of the following sets of axioms should be removed/modified:

	Axiom	Confidence value	Action
1(1)	Penguin ⊑ Bird ⊓ ¬ Fly	1.0	[Remove] [Rewrite]
1(2)	→ ~~Bird ⊑ Animal ⊓ Fly~~	0.64	[Remove] [Rewrite]

Display Impact of Removal
Execute Removed Items
Reset All
Rewrite Axiom: Bird ⊑ Animal ⊓ Fly

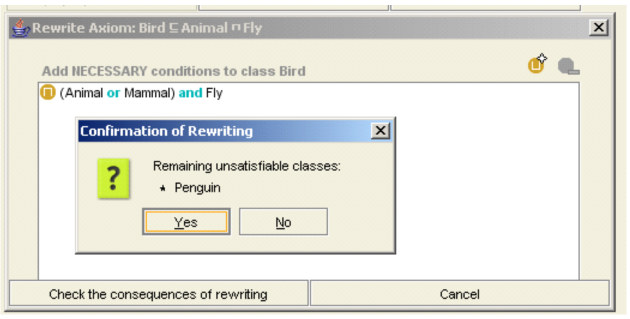
Add NECESSARY conditions to class Bird

(Animal or Mammal) and Fly

Check the consequences of rewriting
Cancel

[18] You can type any axiom to replace the old axiom, and then click `Check the consequences of rewriting`.

[**19**] For demonstration purposes we show an example of a rewrite which will not solve the problem. The system will now prompt that the class Penguin is still unsatisfiable. You can cancel or confirm the rewriting


Rewrite Axiom: Bird ⊑ Animal ⊓ Fly

Add NECESSARY conditions to class Bird

(Animal or Mammal) and Fly

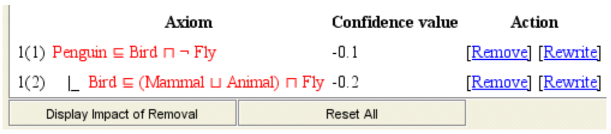
Confirmation of Rewriting

Remaining unsatisfiable classes:
- Penguin

[**20**] If you confirm your rewriting, the plugin will check the consistency of the ontology automatically.

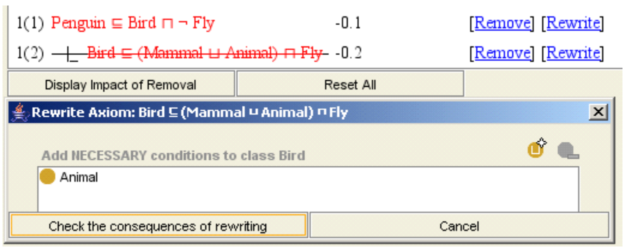
[**21**] The plugin checks the consistency of ontology again.

[**22**] The below figure shows that Penguin is still unsatisfiable.



	Axiom	Confidence value	Action
1(1)	Penguin ⊑ Bird ⊓ ¬ Fly	-0.1	[Remove] [Rewrite]
1(2)	|_ Bird ⊑ (Mammal ⊔ Animal) ⊓ Fly	-0.2	[Remove] [Rewrite]

[**23**] We now try to write the second axiom as Bird ⊑ Animal:



1(1)	Penguin ⊑ Bird ⊓ ¬ Fly	-0.1	[Remove] [Rewrite]
1(2)	~~|_ Bird ⊑ (Mammal ⊔ Animal) ⊓ Fly~~	-0.2	[Remove] [Rewrite]

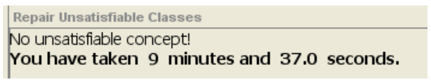
Rewrite Axiom: Bird ⊑ (Mammal ⊔ Animal) ⊓ Fly

Add NECESSARY conditions to class Bird

Animal

Display Impact of Removal
Reset All
Check the consequences of rewriting
Cancel

[**24**] Now, Penguin becomes satisfiable. You can record the time you have taken to do this task.


Repair Unsatisfiable Classes

No unsatisfiable concept!
You have taken 9 minutes and 37.0 seconds.

D.1.2 Exercise: Rewriting Axioms

Now, you have to resolve three ontologies using the plugin.

Note that you should try to preserve the ideas behind the original ontology as much as possible; i.e. do not merely remove axioms to try to get rid of the problems; instead, you should try to rewrite the axioms in a way that preserves whatever the original ontology was trying to capture.

You should use about 20 mins (maximum) for each ontology. If you cannot finish the task, you should save the modified ontologies and record the time you have taken.

Note that: the likely correctness of axioms is evaluated by the system and displayed as the confidence value in this exercise.

Mad_Cows.owl

[**1**] Load the Mad_Cows.owl into Protégé.

[**2**] Activate the RepairTab from the `Configure`··· menu item.

[**3**] Go to the RepairTab, click the icon $?^+$ on the right hand side of the screen, and wait for a while (as the size of ontology is large).

[**4**] Five classes are detected with inconsistency.

[**5**] While you are doing the task, answer the following questions.

Questions:

1. Have you seen this ontology before?

   (a) Yes (b) No (c) Not Sure

2. Do you understand the reasons for the unsatisfiable PartTime_taxi_driver?

   (a) Yes (b) No (c) Not Sure

   If yes, can you state the reason?

3. How do you select the axiom for modification in the unsatisfiable PartTime_taxi_driver?

[**1**] Apply your own background knowledge, without referring to the confidence values.

[**2**] Apply your own background knowledge, and confirm your selection with the confidence values.

[**3**] Select the axiom with the lowest confidence value given by the plugin.

[**4**] Random selection

[**5**] Unknown

4. Do you understand the reasons for the unsatisfiable Sheep?

   (a) Yes (b) No (c) Not Sure

   If yes, can you state the reason?

5. How do you select the axiom for modification in the unsatisfiable Sheep?

[**1**] Apply your own background knowledge, without referring to the confidence values.

[**2**] Apply your own background knowledge, and confirm your selection with the confidence values.

[**3**] Select the axiom with the lowest confidence value given by the plugin.

[**4**] Random selection

[**5**] Unknown

6. Do you think the confidence of axioms given by the plugin useful? Why?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

7. What is your overall comment on the plugin? (any suggestions to make it better?)

8. What is your time taken to do the task?

9. Save your modified ontology

CHEM.owl

[**1**] Load the `CHEM.owl` into Protégé.

[**2**] Activate the RepairTab from the `Configure`··· menu item.

[**3**] Go to the RepairTab, click the icon $?^{+}$ on the right hand side of the screen, and wait for a while (as the size of ontology is large)

[**4**] Thirty-seven classes are detected with inconsistency

[**5**] While you are doing the task, answer the following questions.

Questions:

1. Have you seen this ontology before?
   (a) Yes (b) No (c) Not Sure
2. Do you understand the reasons for the unsatisfiable Person?
   (a) Yes (b) No (c) Not Sure
   If yes, can you state the reason?

3. How do you select the axiom for modification in the unsatisfiable Person?

[**1**] Apply your own background knowledge, without referring to the confidence values.

[**2**] Apply your own background knowledge, and confirm your selection with the confidence values.

[**3**] Select the axiom with the lowest confidence value given by the plugin.

[**4**] Random selection

[**5**] Unknown

4. Do you understand the reasons for the unsatisfiable ChemicalElement?

   (a) Yes (b) No (c) Not Sure

   If yes, can you state the reason?

5. How do you select the axiom for modification in the unsatisfiable ChemicalElement?

   [**1**] Apply your own background knowledge, without referring to the confidence values

   [**2**] Apply your own background knowledge, and confirm your selection with the confidence values

   [**3**] Select the axiom with the lowest confidence value given by the plugin

   [**4**] Random selection

   [**5**] Unknown

6. Do you think the confidence of axioms given by the plugin useful? Why?

   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

7. What is your overall comment on the plugin? (any suggestions to make it better?)

8. What is your time taken to do the task?

9. Save your modified ontology

SUMO-CYC.owl

[**1**] Load the SUMO-CYC.owl into Protégé. Please wait for longer as the size of the ontology is large.

[**2**] Activate the RepairTab from the `Configure`··· menu item.

[**3**] Go to the RepairTab, click the icon ?$^{+}$ on the right hand side of the screen, and wait for a while (as the size of ontology is large).

[**4**] Six classes are detected to be unsatisfiable.

[**5**] While you are doing the task, answer the following questions.

Questions:

1. Have you seen this ontology before?

   (a) Yes (b) No (c) Not Sure

2. Do you understand the reasons for the unsatisfiable Relation?

   (a) Yes (b) No (c) Not Sure

   If yes, can you state the reason?

3. How do you select the axiom for modification in the unsatisfiable Relation?

[**1**] Apply your own background knowledge, without referring to the confidence values.

[**2**] Apply your own background knowledge, and confirm your selection with the confidence values.

[**3**] Select the axiom with the lowest confidence value given by the plugin.

[**4**] Random selection

[**5**] Unknown

4. Do you think the confidence of axioms given by the plugin useful? Why?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

5. What is your overall comment on the plugin? (any suggestions to make it better?)

6. What is your time taken to do the task?

7. Save your modified ontology

Now, you should have five modified ontologies, please send all the ontologies to slam@csd.abdn.ac.uk

Thank you very much!

Appendix E

Evaluation on Fine-grained Approach – Practical Worksheet

E.1 Protégé Plug-in Usability Evaluation – RepairTab

Name	
Date	

E.1.1 Example 1 - Mad Cow Ontology

Background:

[**1**] Mad_cow is detected to be unsatisfiable.

[**2**] Four axioms which cause mad_cow to be unsatisfiable are listed.

[**3**] The parts of axioms which are in blue colour are not relevant to the unsatisfiability.

[**4**] The axioms are ranked based the number of lost relationships if the corresponding axioms are removed.

[**5**] You can remove the whole axiom by selecting the "Remove" link, or remove parts of axioms by clicking on the parts of axiom

[**6**] You can remove the selected axiom directly with the button "Execute Removed Items", or browse the impact of the axiom removal before the axiom is really removed from the ontology, by clicking on the button "Display Impact of Removal".

[**7**] For the selected axioms to be removed

(a) The relationships which will be lost due to the removal are listed.

(b) The helpful changes can be selected to be added back to the ontology.

(c) The harmful changes are the suggestions of changes which should not be added into the ontology.

Answer the following questions:

1. Have you seen this ontology before?

(a) Yes (b) No (c) Not Sure

2. What is your start time?

3. What is your finish time?

4. Do you understand the reasons for the unsatisfiability of mad_cow?
   (a) Yes (b) No (c) Not Sure

5. Do you think the list of problematic axioms which are highlighted with different colours is useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

6. Which axiom do you select to change/remove?

7. For the impact of removed axioms,
   (a) Do you find the list of lost relationships useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

   (b) Do you find the list of helpful changes useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

   If yes, how many helpful changes do you select to add back to the ontology?

   (c) Do you find the list of harmful changes useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

8. Which axiom do you select to change/remove?

9. How many changes do you make on the ontology totally?

10. What is your overall comment on the plug-in ?

E.1.2 Example 2 - Food Ontology

Background:

[**1**] VeganFood is an unsatisfiable concept.

[**2**] The axioms which cause the concept to be unsatisfiable are displayed.

Answer the following questions:

1. Have you seen this ontology before?
   (a) Yes (b) No (c) Not Sure
2. What is your start time?

3. What is your finish time?

4. Do you understand the reasons for the unsatisfiability of mad_cow?
   (a) Yes (b) No (c) Not Sure

5. Do you think the list of problematic axioms which are highlighted with different colours is useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

6. Which axiom do you select to change/remove?

7. For the impact of removed axioms,
   (a) Do you find the list of lost relationships useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

   (b) Do you find the list of helpful changes useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

   If yes, how many helpful changes do you select to add back to the ontology?

   (c) Do you find the list of harmful changes useful?
   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

8. Which axiom do you select to change/remove?

9. How many changes do you make on the ontology totally?

10. What is your overall comment on the plug-in ?

E.1.3 Example 3 - University Ontology

Background:

[**1**] There are 12 unsatisfiable concepts..

[**2**] You are suggested to resolve them based on the order of the list.

[**3**] For each unsatisfiable concept, the axioms which cause the concept to be unsatisfiable are displayed.

Answer the following questions:

1. Have you seen this ontology before?
   (a) Yes (b) No (c) Not Sure
2. What is your start time?

3. What is your finish time?

4. Do you understand the reasons for the unsatisfiability of mad_cow?

   (a) Yes (b) No (c) Not Sure

5. Do you think the list of problematic axioms which are highlighted with different colours is useful?

   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

6. Which axiom do you select to change/remove?

7. For the impact of removed axioms,

   (a) Do you find the list of lost relationships useful?

   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

   (b) Do you find the list of helpful changes useful?

   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

   If yes, how many helpful changes do you select to add back to the ontology?

   (c) Do you find the list of harmful changes useful?

   (5) Very useful (4) Quite useful (3) Neutral (2) Not useful (1) Useless

8. Which axiom do you select to change/remove?

9. How many changes do you make on the ontology totally?

10. What is your overall comment on the plug-in ?

Bibliography

[1] Eugenio Alberdi and Derek H. Sleeman. Retax: a step in the automation of taxonomic revision. *Artificial Intelligence*, 91(2):257–279, 1997.

[2] Eugenio Alberdi, Derek H. Sleeman, and Meg Korpi. Accommodating surprise in taxonomic tasks: the role of expertise. *Cognitive Science*, 24(1):53–91, 2000.

[3] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(6):510–530, 1985.

[4] Nicolas Anquetil, Káthia Oliveira, Márcio Greyck Dias, Marcelo Ramal, and Ricardo Meneses. Knowledge for software maintenance. In *15th International conference on software engineering and knowledge engineering, SEKE'03*, 2003.

[5] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, April 2004.

[6] F. Baader, M. Buchheit, and B. Hollunder. Cardinality Restrictions on Concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[7] F. Baader and W. Nutt. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*.

[8] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artifical Intelligence*, 88(1-2):195–213, 1996.

[9] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[10] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithms. In *PDK '91: Proceedings of the International Workshop on Processing Declarative Knowledge*, pages 67–86, London, UK, 1991. Springer-Verlag.

[11] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14(1):149–180, 1995.

[12] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems or "making KRIS get a move on". In B. Nebel, W. Swartout, and C. Rich, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference*, pages 270–281, San Mateo, 1992. Morgan Kaufmann.

[13] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages

for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday*, number 2605 in Lecture Notes in Artificial Intelligence, pages 228–248. Springer, 2005.

[14] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[15] Kenneth Baclawski, Mieczyslaw M. Kokar, Richard J. Waldinger, and Paul A. Kogut. Consistency checking of semantic web ontologies. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, Lecture Notes in Computer Science, pages 454–459, London, UK, 2002. Springer-Verlag.

[16] Kenneth Baclawski, Christopher J. Matheus, Mieczyslaw M. Kokar, J. Letkowski, and Paul A. Kogut. Towards a symptom ontology for semantic web applications. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference*, Lecture Notes in Computer Science, pages 650–667. Springer, 2004.

[17] Patricia G. Baker, Carole A. Goble, Sean Bechhofer, Norman W. Paton, Robert Stevens, and Andy Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999.

[18] Sean Bechoffer, Frank van Harmlen, Jim Hendler, Ian Horrocks, Deborah McGuinnes, Peter Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference, February 2004. http://www.w3.org/TR/owl-ref/.

[19] Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper: San Francisco, 1999.

[20] Abraham Bernstein and Esther Kaufmann. GINO - a guided input natural language ontology editor. In *International Semantic Web Conference*, pages 144–157, 2006.

[21] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan. Approximation and difference in description logics. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 203–214, San Francisco, CA, 2002. Morgan Kaufman.

[22] Janez Brank, Marko Grobelnik, and Dunja Mladenić. A survey of ontology evaluation techniques. In *Proceedings of Data Mining and Data Warehouses (SiKDD 2005)*, October 2005.

[23] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, February 2004. http://www.w3.org/TR/rdf-schema/.

[24] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[25] Tim Burners-Lee, James Hendler, , and Ora Lassila. The semantic web. *Scientific American*, 284(5), May 2001.

[26] Silvana Castano, Alfio Ferrara, and Gianpaolo Messa. Results of the hmatch ontology matchmaker in OAEI 2006. In *International Workshop on Ontology Matching*, November 2006.

[27] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. Okbc: a programmatic foundation for knowledge base interoperability. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 600–607, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[28] Fellbaum Christiane. *WordNet: An Electronic Lexical Database and Some of Its Applications*. The MIT Press, 1996.

[29] Philipp Cimiano, Aleksander Pivk, Lars Schmidt-Thieme, and Steffen Staab. Learning taxonomic relations from heterogeneous evidence. In P. Buitelaar, P. Cimiano, and B. Magnini, editors, *Ontology Learning from Text: Methods, Applications and Evaluation*, number 123, pages 59–73. IOS Press, 2005.

[30] Philipp Cimiano and Steffen Staab. Learning by googling. *SIGKDD Explor. Newsl.*, 6(2):24–33, 2004.

[31] Philipp Cimiano and Johanna Völker. Text2onto - a framework for ontology learning and data-driven change discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'05)*, JUN 2005.

[32] Ronald Cornet and Ameen Abu-Hanna. Evaluation of a frame-based ontology. a formalization-oriented approach. In R. Engelbrecht G. Surján and P. McNair, editors, *MIE2002, Studies in Health Technology and Information*, volume 90, 2002.

[33] B. Cuenca-Grau, I. Horrocks, O. Kutz, and U. Sattler. Will my ontologies fit together? In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, 2006.

[34] Mukesh Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Seventh National Conference on Artificial Intelligence*, volume 2, pages 475–479, Menlo Park, California, 1988. AAAI Press.

[35] Kevin Dunbar. *Creative thought: An investigation of conceptual structures and processes*.

[36] Marc Ehrig and Steffen Staab. QOM –quick ontology mapping. In *Proceedings of the Third International Semantic Web Conference*, pages 683–697. Springer-Verlag, 2004.

[37] K. Anders Ericsson and Herbert A. Simon, editors. *Protocol analysis: Verbal reports as data*. Cambridge, MA: Bradford Books/MIT Press, 1984.

[38] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: a tool for collaborative ontology construction. *Int. J. Hum.-Comput. Stud.*, 46(6):707–727, 1997.

[39] Edward A. Feigenbaum. The art of artificial intelligence: Themes and case studies of knowledge engineering. In *IJCAI*, pages 1014–1029, 1977.

[40] Giorgos Flouris. *On Belief Change and Ontology Evolution*. PhD thesis, Dept. of Computer Science, University of Crete, 2006.

[41] Giorgos Flouris, Zhisheng Huang, Jeff Z. Pan, Dimitris Plexousakis, and Holger Wache. Inconsistencies, negations and changes in ontologies. In *Proc. of the 21st National Conference on Artificial Intelligence (AAAI-06)*, July 1620 2006. To appear.

[42] Giorgos Flouris and Dimitris Plexousakis. Handling ontology change:. survey and proposal for a future research direction. Technical Report TR-362, Institute of Computer Science, F.O.R.T.H., September 2005.

[43] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. Generalizing the AGM postulates: preliminary results and applications. In *Proc. of the 10th International Workshop on Non-Monotonic Reasoning 2004 (NMR-04)*, pages 171–179, June 2004.

[44] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. On applying the AGM theory to DLs and OWL. In *International Semantic Web Conference 2005*, pages 216–231, 2005.

[45] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. On Applying the AGM Theory to DLs and OWL. In *Proceedings of 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *LNCS*, pages 216–231, November 2005.

[46] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. Evolving ontology evolution. In *SOFtware SEMinar (SOFSEM)*, pages 14–29, 2006.

[47] Marsha E. Fonteyn, Benjamin Kuipers, and Susan J. Grobe. A description of think aloud method and protocol analysis. *Qualitative Health Research*, 3(4):430–441, 1993.

[48] Jon W. Freeman. Hard random 3-sat problems and the davis-putnam procedure. *Artificial Intelligence*, 81(1-2):183–198, 1996.

[49] Gerhard Friedrich and Kostyantyn Shchekotykhin. Diagnosis of description logic knowledge bases. In *Proceedings of 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *LNCS*, November 2005.

[50] Peter Gärdenfors. The dynamics of belief systems: Foundations versus coherence theories. *Revue Internationale de Philosophie 44*, 44.

[51] Peter Gärdenfors. *Belief Revision: An Introduction*. Cambridge University Press, 1992.

[52] S. Ghilardi, C. Lutz, and F. Wolter. Did i damage my ontology? a case for conservative extensions in description logic. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR2006)*, Lake District, UK, 2006.

[53] Fausto Giunchiglia and Roberto Sebastiani. A SAT-based decision procedure for ALC. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 304–314, San Francisco, California, 1996. Morgan Kaufmann.

[54] Asunción Gómez-Pérez, Óscar Corcho, and Mariano Fernandez-Lopez. *Ontological Engineering (Advanced Information and Knowledge Processing)*. Springer, 2003.

[55] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using e-connections. *Journal of Web Semantic*, 4(1):40–59, 2006.

[56] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[57] Nicola Guarino and Christopher Welty. An overview of ontoclean. In *Handbook on Ontologies*, pages 151–159, Germany, 2004. Springer Verlag.

[58] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 403–412. ACM Press, 2004.

[59] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge

bases: A practical case study. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence*, pages 161–168, 2001.

[60] Volker. Haarslev and Ralf Möller. Racer system description. In *International Joint Conference on Automated Reasoning, IJCAR'2001*, 2001.

[61] Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 61–75, London, UK, 2001. Springer-Verlag.

[62] Peter Haase, Jeen Broekstra, Marc Ehrig, Maarten Menken, Peter Mika, Michal Plechawski, Pawel Pyszlak, Björn Schnizler, Ronny Siebes, Steffen Staab, and Christoph Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the Third International Semantic Web Conference*, November 2004.

[63] Peter Haase and Ljiljana Stojanovic. Consistent evolution of OWL ontologies. In *2nd European Semantic Web Conference 2005*, pages 182–197, 2005.

[64] Peter Haase and York Sure. Usage tracking for ontology evolution. SEKT Deliverable 3.2.1, University of Karlsruhe, January 2005. Also available as http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/SEKT-D3.2.1.pdf.

[65] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In *Proc. of the Fourth International Semantic Web Conference (ISWC2005)*. Springer, November 2005.

[66] Christian Halaschek-Wiener, Bijan Parsia, and Evren Sirin. Description logic reasoning with syntactic updates. In *5th international conference on ontologies, databases, and applications of semantics, ODBASE 2006*, 2006.

[67] Adil Hameed, Derek Sleeman, and Alun Preece. Detecting mismatches among experts ontologies acquired through knowledge elicitation. *Knowledge-Based Systems*, 15(05-Jun):265–273, 06 2002.

[68] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[69] Daniel Hewlett, Aditya Kalyanpur, Vladamir Kovlovski, , and Chris Halaschek-Wiener. Effective natural language paraphrasing of ontologies on the semantic web. In *End User Semantic Web Interaction Workshop (ISWC 2005)*, November 2005.

[70] Pascal Hitzler, Markus Krötzsch, Marc Ehrig, and York Sure. What is ontology merging? - a category-theoretical perspective using pushouts. In Alain Leger Deborah L. McGuinness Pavel Shvaiko, Jerome Euzenat and Holger Wache, editors, *Proceedings of the First International Workshop on Contexts and Ontologies: Theory, Practice and Applications*, July 2005.

[71] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR-91*, pages 335–346, Boston (USA), 1991.

[72] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90)*, 1990.

[73] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, , and Hai Wang. The manchester owl syntax. In *OWL: Experiences and Directions 2006*, November 2006.

[74] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[75] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[76] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *2003 International Semantic Web Conference (ISWC 2003)*, pages 17–29. Springer, October 2003.

[77] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.

[78] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[79] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[80] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In David MacAllester, editor, *CADE-2000*, number 1831 in LNAI, pages 482–496. Springer-Verlag, 2000.

[81] Eduard Hovy. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In *Proceedings of the 1st. International Conference on Language Resourcesand Evaluation (LREC)*, 1998.

[82] Wei Hu, Gong Cheng, Dongdong Zheng, Xinyu Zhong, and Yuzhong Qu. The results of Falcon-AO in the OAEI 2006 campaign. In *International Workshop on Ontology Matching*, November 2006.

[83] Zhisheng Huang and Heiner Stuckenschmidt. Reasoning with multi-version ontologies: a temporal logic approach.

[84] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, August 2005.

[85] Ullrich Hustadt and Renate A. Schmidt. On evaluating decision procedures for modal logic. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 202–209. Morgan Kaufmann, 1997.

[86] Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, Dept. of Computer Science, University of Maryland, 2006.

[87] Aditya Kalyanpur, Bijan Parsia, and Bernardo Cuenca-Grau. Beyond asserted axioms: Fine-grain justifications for OWL-DL entailments. In *International Workshop on Description Logics (DL 2006)*, June 2006.

[88] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca Grau, and Evren Sirin. Tableaux tracing

in SHOIN. Technical Report UMIACS-TR 2005-66, University of Maryland, Nov 2005.

[89] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca Grau, and Evren Sirin. Justifications for entailments in expressive description logics. Technical report, University of Maryland, Jan 2006.

[90] Aditya Kalyanpur, Bijan Parsia, and Evren Sirin. Black box techniques for debugging unsatisfiable concepts. In *International Workshop on Description Logics*, 2005.

[91] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca-Grau. Repairing Unsatisfiable Concepts in OWL Ontologies. In *Proceedings of the Third European Semantic Web Conference (ESWC-2006)*, June 2006.

[92] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo C. Grau, and James Hendler. Swoop: A web ontology editing browser. *Journal Of Web Semantics*, 4(2):144–153, June 2006.

[93] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. 3(4), June 2005. Journal of Web Semantics.

[94] Hirofumi Katsuno and Alberto Mendelzon. On the difference between updating a knowledge base and revising it. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 387–394. Morgan Kaufmann, San Mateo, California, 1991.

[95] Hirofumi Katsuno and Alberto O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1992.

[96] Michel Klein and Dieter Fensel. Ontology versioning for the semantic web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, July 2001.

[97] M. Korpi. *Making conceptual connections: an investigation of cognitive strategies and heuristics for inductive categorization with natural concepts*. PhD thesis, Stanford University, 1988.

[98] Joey Sik Chun Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. Ontology inconsistency handling: ranking and rewriting axioms. Technical Report AUCS/TR0603, University of Aberdeen, June 2006.

[99] Joey Sik Chun Lam, Derek Sleeman, and Wamberto Vasconcelos. ReTAX+: a cooperative taxonomy revision tool. In *The Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 64 – 77. Springer, 2004.

[100] Joey Sik Chun Lam, Derek Sleeman, and Wamberto Vasconcelos. Graph-based ontology checking. In *the workshop Ontology Management: Searching, Selection, Ranking, and Segmentation in K-CAP 05*, October 2005.

[101] Joey Sik Chun Lam, Derek Sleeman, and Wamberto Vasconcelos. ReTAX++: a tool for browsing and revising ontologies. In *Poster Proceedings of the 4th International Semantic Web Conference*. Springer, November 2005.

[102] Archana Laxmisana, Sameer Malhotraa, Alla Keselmana, Todd R. Johnsonb, and Vimla L. Patel. Decisions about critical events in device-related scenarios as a function of expertise. *Journal of Biomedical Informatics*, 38(3):200–212, June 2005.

[103] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Software anthropology: Performing field studies in software companies. In *Consortium for Software Engineering Research (CSER)*, 1996.

[104] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[105] Carsten Lutz, Franz Baader, Enrico Franconi, Domenico Lembo, Ralf Möller, Riccardo Rosati, Ulrike Sattler, Boontawee Suntisrivaraporn, and Sergio Tessaris. Reasoning support for ontology design. In *Proceedings of the second international workshop OWL: Experiences and Directions*, November 2006.

[106] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management - EKAW-2002*. LNCS, Springer, October 2002.

[107] Frank Manola and Eric Miller. RDF Primer W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-primer/.

[108] Sabine Massmann, Daniel Engmann, and Erhard Rahm. COMA++: Results for the ontology alignment contest OAEI 2006. In *International Workshop on Ontology Matching*, November 2006.

[109] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. The chimaera ontology environment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 1123–1124. AAAI Press / The MIT Press, 2000.

[110] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[111] Chris Mellish and Xiantang Sun. The semantic web as a linguistic resource: Opportunities for natural language generation. In *the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, December 2005.

[112] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic ALC. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, July 2006.

[113] Chieko Nakabasami and Naoyuki Nomura. A proposal for screening inconsistencies in ontologies based on query languages using wsd. In *NLPXML '02: Proceedings of the 2nd workshop on NLP and XML*, pages 1–5, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[114] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[115] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory, KSL-01-05, March 2001.

[116] Natalya F. Noy and Mark A. Musen. Promptdiff: a fixed-point algorithm for comparing ontology versions. In *Eighteenth national conference on Artificial intelligence*, pages 744–750, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[117] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubezy, Ray W. Fergerson, and Mark A. Musen. Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems*, 2(16):60–71, 2001.

[118] Natalya Fridman Noy, Sandhya Kunnatur, Michel C. A. Klein, and Mark A. Musen. Tracking changes during ontology evolution. In F. van Harmelen (eds) S.A. McIlraith, D. Plexousakis, editor, *Proceedings of the 3rd International Conference on the Semantic Web (ISWC-04)*, volume 3298/2004, pages 379–384. Springer-Verlag.

[119] Natalya Fridman Noy and Mark A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 450–455, 2000.

[120] Daniel Oberle, Raphael Volz, Steffen Staab, and Boris Motik. An extensible ontology software environment. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 299–320. Springer, 2004.

[121] Bijan Parsia, Christian Halaschek-Wiener, and Evren Sirin. Towards incremental reasoning through updates in OWL DL. In *Reasoning on the Web, RoW 2006*, May 2006.

[122] Peter Plessers and Olga De Troyer. Resolving inconsistencies in evolving ontologies. In *Proceedings of the Third European Semantic Web Conference (ESWC-2006)*, June 2006.

[123] A L Rector, S K Bechhofer, C A Goble, I Horrocks, W A Nolan, and W D Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9(2):139171, Feb 1997.

[124] A. L. Rector, N. Horridge, M. Rogers, J. Knublauch, H. Stevens, R. Wang, and C. Wroe. Designing user interfaces to minimise common errors in ontology development: The co-ode and hyontuse projects. In *UK E-Science All Hands Meeting 2004(AHM04)*. ACM Press, Aug 2004.

[125] Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *14th International Conference on Knowledge Engineering and Knowledge Management EKAW 2004*, pages 63–81, January 2004.

[126] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[127] Thomas Russ, Andre Valente, Robert MacGregor, and William Swartout. Practical experiences in trading off ontology usability and reusability. In *Knowledge Acquisition Workshop (KAW99)*, 1999.

[128] Stuart Russell and Peter Norvig, editors. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[129] Satya S. Sahoo, Christopher Thomas, Amit Sheth, William S. York, and Samir Tartir. Knowledge modeling and its application in life sciences: a tale of two ontologies. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 317–326, New York, NY, USA, 2006. ACM Press.

[130] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *ESWC*, pages 226–240, 2005.

[131] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *8th International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.

[132] Stefan Schlobach and Zhisheng Huang. Inconsistent ontology diagnosis: Framwork and prototype. SEKT Deliverable 3.6.1, University of Karlsruhe, October 2005.

[133] Stefan Schlobach, Zhisheng Huang, and Ronald Cornet. Inconsistent ontology diagnosis: Evaluation. SEKT Deliverable 3.6.2, University of Karlsruhe, January 2006.

[134] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artifical Intelligence*, 48(1):1–26, 1991.

[135] Evren Sirin, Bernardo Cuenca Grau, and Bijan Parsia. From wine to water: Optimizing description logic reasoning for nominals. In *International Conference on the Principles of Knowledge Representation and Reasoning (KR-2006)*, pages 90–99. AAAI Press, June 2006.

[136] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*. To appear.

[137] B C Smith, J Williams, and S Schulze-Kremer. The ontology of the gene ontology. In *AMIA Annual Symposium*, 2003.

[138] Marc Spaniol, Ralf Klamma, and Matthias Jarke. Data integration for multimedia e-learning environments with xml and mpeg-7. In *Proceedings of the 4th International Conference on Practical Aspects of Knowledge Management*, pages 244–255, London, UK, 2002. Springer-Verlag.

[139] Steffen Staab, Rudi Studer, Hans-Peter Schnurr, and York Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[140] Mark E. Stickel, Richard J. Waldinger, and Vinay K. Chaudhri. *A Guide to SNARK*. http://www.ai.sri.com/snark/tutorial/tutorial.html.

[141] Giorgos Stoilos, Giorgos B. Stamou, and Stefanos D. Kollias. A string metric for ontology alignment. In *International Semantic Web Conference*, pages 624–637, 2005.

[142] Ljiljana Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe, 2004.

[143] Gerd Stumme and Alexander Maedche. Ontology merging for federated ontologies on the semantic web. In *Proceedings of the International Workshop for Foundations of Models for Information Integration (FMII-2001)*, September 2001.

[144] Julien Tane, Christoph Schmitz, and Gerd Stumme. Semantic resource management for the web: an e-learning application. In *Proceedings of the 13th international World Wide Web conference*, pages 1–10, New York, NY, USA, 2004. ACM Press.

[145] Gunnar Teege. Making the difference: A subtraction operation for description logics. In *the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR94)*, 1994.

[146] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[147] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.

[148] Mike Uschold, Mike Healy, Keith Williamson, Peter Clark, and Steven Woods. Ontology reuse and application. In *Formal Ontology in Information Systems*. ISO Press, Amsterdam,

1998.

[149] Maarten v. Someren, Yvonne F. Barnard, and Jacobijn A.C. Sandberg, editors. *The Think Aloud Method: A practical guide to modelling cognitive processes*. Academic Press, New York, 1994.

[150] Willem van Hage, Sophia Katrenko, and Guus Schreiber. A method to combine linguistic ontology-mapping techniques. In *4th International Semantic Web Conference*, pages 732–744. Springer-Verlag, 2005.

[151] Willem van Hage, Sophia Katrenko, and Guus Schreiber. A method to combine linguistic ontology-mapping techniques. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*, pages 732–744. Springer Verlag Berlin-Heidelberg, NOV 2005.

[152] Johanna Völker, Denny Vrandecic, and York Sure. Automatic evaluation of ontologies (AEON). In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*, pages 716–731. Springer Verlag Berlin-Heidelberg, NOV 2005.

[153] Denny Vrandecic. Explicit knowledge engineering patterns with macros. In *Chris Welty and Aldo Gangemi, Proceedings of the Ontology Patterns for the Semantic Web Workshop at the ISWC 2005*, November 2005.

[154] Hai Wang, Matthew Horridge, Alan Rector, Nick Drummond, and Julian Seidenberg. Debugging OWL-DL Ontologies: A heuristic approach. In *4th Internation Semantic Web Conference*, 2005.

[155] Christopher Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data Knowledge Engineering*, 39(1):51–74, 2001.

[156] Graham Wilcock. Talking OWLs: Towards an ontology verbalizer. In *Human Language Technology for the Semantic Web and Web Services, ISWC'03*, pages 109–112, November 2003.

[157] Jin Yu, Ehud Reiter, Jim Hunter, and Somayajulu Sripada. SUMTIME-TURBINE: A knowledge-based system to communicate time series data in the gas turbine domain. In *P Chung et al (Eds) Developments in Applied Artificial Intelligenc: Proceedings of IEA/AIE-2003*, volume 2718, pages 379–384. Springer, 2003.