THE JUSTIFICATORY STRUCTURE OF OWL ONTOLOGIES

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2013

By
Samantha Patricia Bail
School of Computer Science

Contents

Word Count: 54,030

List of Tables

List of Figures

Abstract

The Web Ontology Language OWL is based on the highly expressive description logic $\mathcal{SROIQ}$, which allows OWL ontology users to employ out-of-the-box reasoners to compute information that is not only explicitly asserted, but *entailed* by the ontology. *Explanation* facilities for entailments of OWL ontologies form an essential part of ontology development tools, as they support users in detecting and repairing errors in potentially large and highly complex ontologies, thus helping to ensure ontology quality.

Justifications, minimal subsets of an ontology that are sufficient for an entailment to hold, are currently the prevalent form of explanation in OWL ontology development tools, and they have been found to significantly reduce the time and effort required to debug erroneous entailments. A large number of entailments, however, have not only one but *many* justifications, which can make it considerably more challenging for a user to find a suitable repair for the entailment.

In this thesis, we investigate the relationships between multiple justifications for both single and multiple entailments, with the goal of exploiting this *justificatory structure* in order to devise new *coping* strategies for multiple justifications. We describe various aspects of the justificatory structure of OWL ontologies, such as shared axiom cores and structural similarities. We introduce a model for measuring user effort in the debugging process and propose debugging strategies that exploit the justificatory structure in order to reduce user effort. Finally, an analysis of a large corpus of ontologies from the biomedical domain reveals that OWL ontologies used in practice frequently exhibit a rich justificatory structure.

Date of submission: 26/04/2013

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Chapter 1

Introduction

Since its standardisation by the W3C in 2004, the Web Ontology Language OWL [W3C09] has become a widely used language for representing ontological knowledge. OWL ontologies are used across a wide spectrum of domains, ranging from chemistry to bio-health informatics and medical data. In its latest version, OWL 2 [CHM+08], which is based on the expressive description logic $\mathcal{SROIQ}$ [BCM+03, HKS06], allows ontology modellers to use highly complex expressions to describe the entities in an ontology.

An OWL ontology is a set of *axioms*, which make statements about the classes, properties, and individuals in an ontology, as well as the relations between them. Due to its foundations in description logics [BCM+03] and the availability of highly optimised description logic reasoners, OWL ontology modellers and users can apply automated reasoning techniques to elicit not only explicitly asserted knowledge in the ontology, but also *implicit* information which is *entailed* by the ontology. This makes it possible to model complex hierarchical relations without the need to make explicit every single relation between sub- and super-classes. For example, an ontology which contains the axioms $\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}$ ('every Cat is a Carnivore ') and $\mathsf{Carnivore} \sqsubseteq \mathsf{Animal}$ ('every Carnivore is an Animal ') also entails the statement $\mathsf{Cat} \sqsubseteq \mathsf{Animal}$, without having to explicitly state it.

Ontology development often resembles traditional software development: from the first design to a release candidate, axioms and entities are frequently added, modified, and removed. Some OWL ontologies, such as the National Cancer Institute (NCI) Thesaurus ontology [dCHS+04], undergo a highly dynamic iterative development process, with a team of developers working to produce monthly updates of the ontology. As ontology re-use and sharing on the web is highly encouraged, existing OWL ontologies may be imported, or merged into newly built ones. OWL editing tools, such as the Protégé 4 editor,[1] allow ontology developers to create large and complex ontologies; we often find OWL ontologies

[1] `http://protege.stanford.edu/`

that contain thousands of classes and axioms which are highly interlinked across the ontology [WPH06, HPS11].

All these processes can lead to *errors* in the resulting ontology. The two classes of commonly occurring errors in OWL ontologies are unwanted entailments and unwanted *non*-entailments. Among the former, *incoherence* and *inconsistency* of an ontology can be detected automatically, as they have distinguishing logical features while also being reliably connected to errors. Incoherence refers to the presence of an *unsatisfiable* class in the ontology, that is, a class which cannot have any instances, while *inconsistency* of the ontology means that the ontology does not have any meaningful entailments at all. Similar to the debugging process in software development, the *repair* of such errors involves finding those parts of the ontology that cause the problem, then modifying or removing them in order to rectify the error.

Due to the large size and often complex structure of OWL ontologies, finding and repairing these errors without appropriate tool support can be daunting and error-prone: simply removing those parts of the ontology which are suspected to cause the error may result in a significant loss of *relevant* information but is not guaranteed to sufficiently repair the ontology. This is where *explanation* comes into play: on an abstract level, an explanation for an entailment of an ontology is a statement, or a set of statements, which traces the source of the error and *explains* to the user *what* information in the ontology lead to the entailment and *how* it does this.

Explanation has a long history in the research on intelligent systems (e.g. [Cla81, BS84a]). It has been considered an important component of expert systems, such as MYCIN [BS84b], since the early 1980s, not just for the purpose of *debugging* a system, but also for tasks such as learning about the information modelled in the system, and convincing human experts of the system's correctness. Because of its role in these tasks, explanation is thought to enhance both the correctness and the acceptance of an intelligent system. Russell and Norvig [RN03] illustrate this point with an example of the role explanation facilities play in user-facing expert systems:

> *'A leading expert on lymph-node pathology describes a fiendishly difficult case to the expert system, and examines the system's diagnosis. He scoffs at the system's response. Only slightly worried, the creators of the system suggest he ask the computer for an explanation of the*

diagnosis. The machine points out the major factors influencing its decision and explains the subtle interaction of several of the symptoms in this case. The experts admits his error, eventually.'

Explanation can assume various forms, such as a trace of rules in rule-based expert systems [BS84b], or proofs in logic-based systems [BFH99, McG96]. In OWL tools, explanation generally takes the form of a *pinpoint* of the set—or sets—of axioms that cause the entailment to hold, which we commonly denote as *justifications*.

1.1 Justification-based debugging support

Justification-based explanation support for description logic ontologies is the most prevalent form of explanation in current OWL development tools, with OWL editors such as Protégé 4 and Swoop[2] providing justification-based debugging tools. There exists a large body of research from the past decade which underlines the important role justification-based explanations play in the ontology development process (e.g. [SC03, KPSH05, Lam07, HPS08]).

A justification $\mathcal{J}$ for an entailment η of an ontology $\mathcal{O}$ is a minimal subset of $\mathcal{O}$ which is sufficient for η to hold. Justifications have two major benefits: first, the minimality of a justification prevents the tedious process of searching through the ontology in order to find the responsible axioms, while also allowing users to focus on a (usually) small subset of the ontology. And second, since a justification simply consists of axioms occurring in $\mathcal{O}$, ontology developers do not have to learn any additional mechanisms or notations in order to understand the explanation. The following example illustrates the idea of justification-based debugging support:

Example 1.1.

(1) $\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}$
(2) $\mathsf{Carnivore} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}$
(3) $\mathsf{Plant} \sqsubseteq \neg \mathsf{Animal}$
(4) $\mathsf{Grass} \sqsubseteq \mathsf{Plant}$
(5) $\mathsf{PetOwner} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasPet}.\mathsf{Animal}$
(6) $\mathsf{Cat} \sqsubseteq \exists \mathsf{eats}.\mathsf{Mouse}$
(7) $\mathsf{SickCat} \equiv \mathsf{Cat} \sqcap \exists \mathsf{eats}.\mathsf{Grass}$
(8) $\exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal}$

[2] `https://code.google.com/p/swoop/`

The ontology $\mathcal{O}$ comprising the axioms in the above example makes statements about the entities in the domain of animals and their eating habits.[3] Several other statements follow logically from $\mathcal{O}$, for example $\mathsf{Cat} \sqsubseteq \mathsf{Animal}$ ('everything that is a Cat is an Animal') and the unsatisfiability of the concept SickCat, which is expressed as $\mathsf{SickCat} \sqsubseteq \bot$. In ontology development tools, such unsatisfiable concepts are usually highlighted to make the user aware of them, for example by rendering them in red colour, or arranging them as a sub-concept of Nothing.

Unsatisfiable classes are commonly regarded as errors, as they cannot have any instances. An ontology developer would want to repair the error by re-writing or removing some of the axioms responsible for causing it. Without a debugging tool, the user has to browse the ontology, find the responsible axioms, and apply modifications they consider appropriate. Providing a justification for the entailment, however, allows the user to focus directly on the responsible axioms, thus clearly reducing the effort involved in finding a suitable repair—and preventing the user from going down the 'wrong path' or giving up their search for the 'needle in the haystack'.

In our example ontology, there exists exactly one justification for the entailment $\mathsf{SickCat} \sqsubseteq \bot$, namely the set consisting of the axioms $\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}$, $\mathsf{Carnivore} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}$, $\mathsf{Grass} \sqsubseteq \mathsf{Plant}$, $\mathsf{SickCat} \equiv \mathsf{Cat} \sqcap \exists \mathsf{eats}.\mathsf{Grass}$, and $\mathsf{Plant} \sqsubseteq \neg \mathsf{Animal}$. The contradiction stems from the fact that Cat is asserted to be an animal which only eats other animals, but SickCat eats Grass, which is asserted to be a Plant, thus not an Animal. There are different approaches to repairing this error, such as weakening the restriction that a Carnivore eats *only* Animals; however, as we can already see from this small example, finding an appropriate repair without weakening or sacrificing too much information is a non-trivial task and requires the user to understand the relations between the axioms in the justification.

1.1.1 Understanding justifications

Repairing an entailment becomes more challenging as the number of justifications for an entailment grows. Given the entailment $\mathsf{Cat} \sqsubseteq \mathsf{Animal}$ from the above ontology in example, we can find two justifications: the set consisting of the two axioms $\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}$ and $\mathsf{Carnivore} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}$, and the second, perhaps less obvious, justification containing $\mathsf{Cat} \sqsubseteq \exists \mathsf{eats}.\mathsf{Mouse}$ and $\exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal}$. A

[3] For the purpose of this example, we assume the statement $\exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal}$ to be correct.

user who wants to understand why $\mathsf{Cat} \sqsubseteq \mathsf{Animal}$ follows from the ontology will be presented with both justifications; if this entailment was unwanted, the user would have to understand and break *both* justifications in order to remove it from the ontology.

Assume the user is faced not only with a small toy ontology as the one above, but with a large and complex ontology, containing several thousand classes, properties, individuals, and axioms. The number of justifications for a single entailment can grow exponentially in the number of axioms in the ontology, with several thousand found in some ontologies used in practice [BHPS11]. And still, if the user wants to ensure the entailment does no longer hold in the ontology, every single justification has to be broken. While it is possible for a user to inspect and modify every justification in isolation, this approach is time-consuming and error-prone. What we need is a strategy that helps users *cope* with multiple justifications and supports them in finding an appropriate repair.

The question of how users *interact* with justifications once they have been computed has been receiving some attention over the past few years [KPC06, LPSV06, HPS10b, HBPS11a]. Depending on the respective task, whether the explanation is requested to fix an error or to aid understanding of an entailment, this interaction can assume different shapes and objectives. For the purpose of debugging, users may regard a *minimal repair* as crucial, that is, removing an error while losing as little other information as possible. This may be achieved by modifying the given justifications to make them easier to understand to users, or, when dealing with multiple justifications, by identifying axioms which occur across several justifications.

In case of the former, *fine-grained* justifications [LPSV06, KPC06, HPS08] are variants of justifications which are as weak as possible and do not contain any superfluous expressions that could possibly distract a user from the actual cause of the entailment. Such fine-grained justifications have been defined more precisely as *laconic* and *precise* justifications [HPS08], with suitable and efficient algorithms to compute them for entailments of OWL ontologies. In the above example, the justification $\{\mathsf{Cat} \sqsubseteq \mathsf{Carnivore},\ \mathsf{Carnivore} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}\}$ for the entailment $\mathsf{Cat} \sqsubseteq \mathsf{Animal}$ can be re-written into its laconic version $\{\mathsf{Cat} \sqsubseteq \mathsf{Carnivore},\ \mathsf{Carnivore} \sqsubseteq \mathsf{Animal}\}$, which reduces the second axiom to its *relevant* parts. While fine-grained justifications might make it easier to understand a single justification, they do not support understanding multiple justifications for an entailment.

Dealing with multiple justifications, either for individual or for *multiple entailments*, has been somewhat facilitated by the introduction of *root and derived* justifications [PSK05] which assist in the repair of multiple unsatisfiable classes: a root justification is a subset of a derived justification; fixing the root justification also repairs all those justifications which are derived from it. As an example, assume we add the axiom $\mathsf{SickCatOwner} \equiv \mathsf{PetOwner} \sqcap \exists \mathsf{hasPet}.\mathsf{SickCat}$ to the ontology. The class SickCatOwner will then be unsatisfiable, but its unsatisfiability depends entirely on the unsatisfiabilty of the class SickCat. Thus, SickCatOwner is a derived unsatisfiable class, whereas SickCat is the root cause. Root and derived justifications, as used in the ontology editor Swoop, have been successfully shown to reduce user effort when debugging multiple entailments [KPSH05].

1.1.2 Justificatory structure

The debugging tool in the Swoop editor by Kalyanpur was the first attempt to make use of the *relations* between justifications in order to support the user in finding a suitable repair for multiple justifications. In addition to root and derived information, the repair tool also presents metrics for the axioms in the justifications, such as the axiom frequency (the number of justifications an axiom occurs in), impact (the number of entailments an axiom affects), and usage (of terms occurring in the axiom). Based on these metrics, the tool computes a rank for each axiom, with the lowest ranked axioms being suggested for removal or modification. This provides users with important guidance on where to start repairing a set of justifications.

This approach makes use of the fact that there exist structural relations between justifications, such as shared axioms of various extents, ranging from single axioms to subset relationships, as in the case of root and derived justifications. In other words, given a set of entailments of an ontology and their justifications, we can identify the *justificatory structure* of the ontology, that is, the set of features and relations of its justifications.

Despite the potential usefulness for improving debugging support for multiple justifications and the insights we can gain into the relationships between entailments of OWL ontologies, their justifications, and the axioms they contain, there has been no further exploration of the justificatory structure of OWL ontologies. Therefore, one of the main goals of this thesis is to identify the different aspects of justificatory structure of ontologies, to investigate potential applications of these

relations in the debugging process, and to establish how prevalent these structural aspects are in OWL ontologies used in practice.

Shared axiom relationships, such as root and derived justifications, are only one aspect of justificatory structure. Another interesting phenomenon we often find in OWL justifications is the *similarity* between justifications, as we can see in the well-known *Pizza* tutorial[4] ontology: take, for example, the entailment $\mathsf{Fiorentina} \sqsubseteq \mathsf{InterestingPizza}$, which has over 200 justifications. The following example shows three of those justifications; Fiorentina is abbreviated to Fi, hasTopping to hasTop, InterestingPizza to IP, and $\top$ is the description logic notation for the top concept Thing which stands for 'any element in the domain'.

Example 1.2.

(1)	$\mathsf{Fi} \sqsubseteq \mathsf{NamedPizza}$	$\mathsf{Fi} \sqsubseteq \mathsf{NamedPizza}$	
(2)	$\mathsf{NamedPizza} \sqsubseteq \mathsf{Pizza}$	$\mathsf{NamedPizza} \sqsubseteq \mathsf{Pizza}$	$\mathsf{domain}(\mathsf{hasTop}, \mathsf{Pizza})$
(3)	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Tomato}$	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Tomato}$	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Tomato}$
(4)	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Olive}$	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Mozzarella}$	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Garlic}$
(5)	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Spinach}$	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Olive}$	$\mathsf{Fi} \sqsubseteq \exists \mathsf{hasTop}.\mathsf{Spinach}$
(6)	$\mathsf{Spinach} \sqsubseteq \neg \mathsf{Tomato}$	$\mathsf{Mozzarella} \sqsubseteq \neg \mathsf{Tomato}$	$\mathsf{Spinach} \sqsubseteq \neg \mathsf{Tomato}$
(7)	$\mathsf{Spinach} \sqsubseteq \neg \mathsf{Olive}$	$\mathsf{Mozzarella} \sqsubseteq \neg \mathsf{Olive}$	$\mathsf{Spinach} \sqsubseteq \neg \mathsf{Garlic}$
(8)	$\mathsf{Olive} \sqsubseteq \neg \mathsf{Tomato}$	$\mathsf{Olive} \sqsubseteq \neg \mathsf{Tomato}$	$\mathsf{Garlic} \sqsubseteq \neg \mathsf{Tomato}$
(9)	$\mathsf{IP} \equiv \mathsf{Pizza} \sqcap \geq 3\mathsf{hasTop}.\top$	$\mathsf{IP} \equiv \mathsf{Pizza} \sqcap \geq 3\mathsf{hasTop}.\top$	$\mathsf{IP} \equiv \mathsf{Pizza} \sqcap \geq 3\mathsf{hasTop}.\top$

While these justifications may look complicated at first, they can all be easily summarised as follows:

- Axioms 1-2 (2-3 in the last justification): Fiorentina is a Pizza.
- Axioms 3-5: Fiorentina has three kinds of toppings.
- Axioms 6-8: These three toppings are pairwise disjoint, that is, no element in the domain can be a member of several toppings at the same time.
- Axiom 9: Anything that is a Pizza and has at least three toppings is an InterestingPizza.

Despite the fact that the justifications contain different axioms, we can immediately see that they are very similar and that it suffices to understand the reasoning in the abstract explanation in order to understand the concrete examples. Since Fiorentina is defined to have six different toppings, the justifications for $\mathsf{Fiorentina} \sqsubseteq \mathsf{InterestingPizza}$ are all combinations of three toppings, variations

[4] `http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/`

on the reasons why these toppings are pairwise disjoint, and variations of the reasons why Fiorentina is a Pizza.

This adds another goal to our investigation of justificatory structure: to determine a set of equivalence relations that allow us to identify such structural similarities, and to investigate how prevalent these similarities are in OWL ontologies—or whether the *Pizza* ontology is just a unique case. The presence of structural similarity provides us with a starting point for lifting justifications from their material form to an abstract explanation *template*, which could drastically reduce the cost of understanding the logical structure of each individual justification. This, in turn, reduces the overall effort a user has to apply in order to debug an entailment with multiple structurally similar justifications.

1.1.3 Beyond debugging and repair

While we put strong emphasis on the debugging aspect of justifications, we may also consider their suitability for other purposes in the ontology development process. In the same way as explanation in expert system serves multiple purposes, we can use justifications in OWL ontologies for tasks beyond debugging, such as ontology comprehension and exploration, ontology learning, and generating ontology metrics.

Ontology comprehension [Kee07, BSP09] describes the process of a person understanding what knowledge is modelled in an ontology and how it is modelled; this is of particular relevance to users attempting to use or integrate an existing ontology. We believe that examining a set of entailments and their justifications can help users gain insights into the relations between the pieces of information contained in the ontology, thus enhancing their understanding of the ontology. Going back to the *Pizza* example above, we can see that by inspecting only a small number of justifications and their abstract template, we already have some understanding of the basic reasoning behind *all* classes in the ontology which are entailed to be an InterestingPizza.

Another potentially useful application of justifications is *ontology metrics*: ontology editors commonly display a set of metrics of an ontology, such as the number and types of axioms, the number of entailments, and the logical expressivity based on the constructors used in the ontology. These ontology metrics allow users to infer information about the size and complexity of the ontology, its *richness* of modelling (that is, the level of detail used to describe a class),

and even its general quality [TAM+05, AB06]. Justification-based metrics, such as the numbers of justifications per entailments, the relations between them, and their structural similarities, add another layer to the existing suite of ontology metrics. These metrics can be used to compare the quality, uniformity, and 'interestingness' of ontologies based on their modelling.

1.2 Research objectives

In summary, the main goals of this thesis are to explore the notion of 'justificatory structure' of OWL ontologies, to investigate the landscape of justificatory structure in ontologies used in practice, and to suggest approaches to using the justificatory structure in order to reduce user effort in the ontology debugging process.

First, we aim to gain a clearer understanding of the landscape of justifications in OWL ontologies. Our goal is to determine how prevalent multiple justifications are in OWL ontologies and what shapes (size, axiom types) these justifications commonly assume. We want to identify a set of relations between justifications which go beyond root and derived unsatisfiable classes, taking into account both shared axioms and shared axiom sets between justifications.

This includes structural similarities between justifications: we want to find a (reasonable) set of equivalence relations that allow us to capture similarities between justifications and group them based on their abstract explanation templates. Taking these relations, we want to know whether they commonly occur in OWL ontologies found 'in the wild' and how prevalent they are in order to see how effectively they can be used to *summarise* multiple justifications.

Second, we look at justificatory structure in order to address the problem of coping with multiple justifications in a debugging and repair scenario. We first need to pin down the basic notion of *effort* required to solve a debugging task involving multiple justifications before we can identify ways of exploiting the justificatory structure in order to reduce the effort ontology users have to spend when debugging one or multiple entailments.

1.3 Contributions

- We provide a set of design decisions for computing finite sets of entailments of OWL ontologies and discuss the benefits and drawbacks of the various options. This has practical implications for the display of *selected entailments* in OWL editors and for the use of entailments as a measure of the information content in ontologies. It also informs the generation and analysis of justifications for finite entailment sets in the context of this thesis.
- We introduce the notion of the justificatory structure of OWL ontologies which allows us to characterise structural relations between justifications for both single and multiple entailments. This includes the characterisation of various aspects of justificatory structure, such as justification properties, axiom properties, and axiom overlap between justifications.
- We introduce two equivalence relations between justifications, subexpression-isomorphism and lemma-isomorphism, which are used to analyse the logical diversity of a set of justifications. These equivalence relations partition a set of justifications, with equivalent justifications following the same lines of reasoning. This provides the foundations for representing a set of justifications by an abstract justification *template* which summarises a set of justification, thus reducing the effort involved in understanding multiple justifications.
- We pin down the notion of a *debugging problem* and present a model of measuring *effort* for successfully solving debugging problems using justifications. We then propose strategies to exploit the justificatory structure of a set of justifications in order to reduce user effort when dealing with multiple justifications.
- Finally, we present the results of a survey of the justificatory structure of OWL ontologies from the NCBO Bioportal. We find that a large number of OWL ontologies in the test corpus exhibit a very rich justificatory structure, with frequently occurring multiple justifications, high extents of overlaps, and high-frequency axioms. Using the newly introduced equivalence relations, we find that the logical diversity of OWL justifications is significantly lower than their numbers may indicate, with justification sets being reduced to only 10% of their original size.

1.4 Published Work

Some of the work presented in this thesis has been published at peer-reviewed conferences and workshops. The following publications correspond roughly to chapters 3 (reference 1), 4 (reference 2-3), 5 (reference 4-5), and 6 (reference 6-8) in this thesis:

1. [BPS11] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Extracting finite sets of entailments from OWL ontologies. In *Proceedings of the 24th International Workshop on Description Logics (DL-11)*, 2011.
2. [BPS10b] Samantha Bail, Bijan Parsia, and Ulrike Sattler. The justificatory structure of OWL ontologies. In *Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED-10)*, 2010.
3. [BHPS11] Samantha Bail, Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The justificatory structure of the NCBO Bioportal ontologies. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*, 2011.
4. [BPS12a] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Diversity of reason: Equivalence relations over description logic explanations. In *Proceedings of the 25th International Workshop on Description Logics (DL-12)*, 2012.
5. [BPS12b] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Declutter Your Justifications: Determining Similarity Between OWL Explanations. In *Proceedings of the First International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM-12)*, 2012.
6. [HBPS11b] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In *Proceedings of the 24th International Workshop on Description Logics (DL-11)*, 2011.
7. [HBPS11a] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*, 2011.
8. [HBPS13] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. Toward cognitive support for OWL justifications. *Submitted to: Knowledge-Based Systems*, 2013.

Chapter 2

Background and related work

Knowledge representation (KR) is an area of artificial intelligence research which deals with representing knowledge using symbols, thus allowing the application of automated reasoning techniques to infer new knowledge from given knowledge [BL04]. KR forms the basis of *knowledge-based systems*, 'intelligent' systems which make use of a *knowledge base* (KB) that contains told facts about some domain, as well as procedures to reason over these facts and infer implicit knowledge. While there exists a wide range of approaches to implementing knowledge-based systems, such as logic programming [BG94], frames [Min74], and semantic networks [Sow87], in this thesis we are dealing with description logics (DLs) [BCM+03] as the underlying knowledge representation formalism for knowledge bases. DLs are a family of logics based on a guarded fragment of first-order logic which is more expressive than propositional logic, while still being decidable [BCM+03].

This chapter introduces the basic concepts of description logics, which underpin the Web Ontology Language OWL [HPSv03, CHM+08]. It outlines the syntax and semantics of description logics and fixes relevant notions such as *axioms* and *entailments* of an OWL ontology. It also discusses the landscape of logical and non-logical errors occurring in OWL ontologies, which motivates the need for tailored debugging support. It then introduces *justifications* as an explanation service for entailments of OWL ontologies, and reviews the literature dealing with justification-based explanation. This covers approaches to computing single and multiple justifications, as well as the issues of understanding justifications, justification-based repair of errors, and coping with multiple justifications, which is the main focus of this thesis.

2.1 Description logic knowledge bases

In the first part of this chapter, we will give an introduction to the basic concepts of description logics, such as the description logic syntax and semantics, and the standard reasoning services used with description logic knowledge bases. We will then discuss the relationship between description logics and the Web Ontology Language OWL, and give a brief overview of the different applications of OWL.

2.1.1 DL syntax and semantics

DL syntax

The main building blocks of DLs are atomic *concepts*, *roles*, and *individuals*. With respect to their relationship with first-order logic (FOL), concepts correspond to unary predicates in FOL, roles to binary predicates, and individuals to constants. These entities are used to create more expressive concept and role expressions with the help of constructors, whereby the available constructors depend on the expressivity of the respective DL. As a convention, we will be using the upper-case letters A and B for atomic concepts and C, D, ... for possibly complex concepts; the lower-case letters r, s, ... for role names, and the lower-case letters a, b, ... for individuals.

The basic description logic $\mathcal{ALC}$ ('Attribute Logic with Complement') [SS91], which many other more expressive DLs build upon, allows concept expressions as defined by the following grammar:

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

Given a concept expression C, the *modal depth* of the expression is the maximum nesting depth of constructors in C. The *length* of an expression is the number of occurrences of concept, role, and individual names, as well as constructors. For example, the expression $A \sqcap \forall r.(\exists s.(B \sqcap C))$ has a modal depth of two and a length of nine.

Axioms

A description logic knowledge base $\mathcal{K}$ is generally regarded as a finite set of axioms which are *asserted* in the KB. Axioms are sentences that make statements about

the domain knowledge modelled in the KB. The axioms in a KB are classified into the sets of *TBox*, *RBox* and *ABox* axioms which are denoted as $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{A}$, respectively: $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$. The *signature* $sig(\mathcal{K})$ of a knowledge base $\mathcal{K}$ is the set of all concept, role, and individual names occurring in $\mathcal{K}$.

A TBox axiom α is either a *subsumption* ($C \sqsubseteq D$) or *equivalence* ($C \equiv D$) between two (possibly complex) concepts C and D in a knowledge base. A subsumption axiom expresses that C is a sub-concept of D; that is, every instance of C is also an instance of D. We can say that C 'is-a' D. Equivalence axioms state that two concepts C and D are equivalent, which corresponds to (and is a shorter notation for) a bi-directional subsumption $C \sqsubseteq D$ and $D \sqsubseteq C$.

Subsumption and equivalence are also possible between roles, which are described by RBox axioms:[1] $r \sqsubseteq s$ specifies that r is a sub-role of s, which means that every two individuals that have an r-relationship also have an s-relationship between them. $r \equiv s$ expresses that the two roles are equivalent. Further, we can specify axioms containing role *chains* $r \circ s \sqsubseteq t$, which states that if an individual a has an r-successor b, and b has an s-successor c, then it also holds that a has a t-relationship with c.

ABox axioms make statements about the relations between individuals and concepts, and between individuals and roles: $C(a)$ expresses that the individual a is an instance of the concept C, and $r(a, b)$ specifies that there exists an r-relationship between the individuals a and b.

As an example, we use a small knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ that describes the eating habits of animals on an abstract level:

Example 2.1.

$$\begin{aligned}
\mathcal{T} = \{&\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}, \mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}, \\
&\mathsf{Plant} \sqsubseteq \neg \mathsf{Animal}, \mathsf{Grass} \sqsubseteq \mathsf{Plant} \\
&\mathsf{PetOwner} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasPet}.\mathsf{Animal}, \exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal}\} \\
\mathcal{A} = \{&\mathsf{Cat}(\mathsf{Molly}), \mathsf{Human}(\mathsf{Alice}), \mathsf{hasPet}(\mathsf{Alice}, \mathsf{Molly})\}
\end{aligned}$$

The TBox of this example KB consists of axioms that make statements about the concepts in the domain: cats are carnivores, a carnivore is an animal which only eats animals, plants are disjoint with animals (i.e. no one thing can be a

[1] Note that the use of RBox axioms as described here already goes beyond the basic description logic $\mathcal{ALC}$.

plant and an animal at the same time), grass is a plant, a pet owner is a human who has an animal as a pet, and everything that eats something is an animal. The ABox contains statements about individuals (Molly is a Cat, Alice is a Human), and the relationship between them (Molly is Alice's pet).

A TBox is called *acyclic* if it does not contain any axioms or chains of axioms where an entity occurs in both the right- and the left-hand side of a subsumption or equivalence. That is, an axiom of the type $C \sqsubseteq \exists r.C$ would cause a TBox to be cyclic. Axioms containing only atomic concepts on the left-hand side are called *definitions*, and an *acyclic* TBox containing only definitions where all concepts on the left-hand side have unique names is called a *definitorial* TBox. *General* TBoxes may contain *general concept inclusion* (GCI) axioms, which allow complex concept expressions on both the right- and the left-hand side, such as $\exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal}$.

Naming conventions

The name of a description logic is generally comprised of mnemonics representing the available constructors and axiom types in the respective logic: the letters $\mathcal{N}$ and $\mathcal{Q}$ stand for unqualified ($\geq nr$, $\leq nr$) and qualified number restrictions ($\geq nr.C$, $\leq nr.C$), respectively (for $n \in \mathbb{N}$, r a role), $\mathcal{F}$ represents the functionality of roles ($\geq nr$), $\mathcal{H}$ stands for role hierarchies ($r \sqsubseteq s$), $\mathcal{I}$ is the role inverse r^-, $\mathcal{R}$ stands for complex role inclusions of the type $r \circ s \sqsubseteq r$. The letter $\mathcal{O}$ denotes the presence of nominals, which allows the use of individuals in the place of concepts in TBox axioms: $C \sqsubseteq \{a\}$.

Some notable description logics, besides the aforementioned $\mathcal{ALC}$, include $\mathcal{S}$ [HST99] which corresponds to $\mathcal{ALC}+$ ($\mathcal{ALC}$ plus role transitivity), the less expressive $\mathcal{EL}$ [Bra04] which allows existential quantifiers and intersection, $\mathcal{EL}^{++}$ [BBL05] which corresponds to $\mathcal{EL}$ plus complex role inclusions and nominals, and the highly expressive logics $\mathcal{SHOIN}(D)$ [HPSv03] and $\mathcal{SROIQ}(D)$ [HKS06] which underpin OWL and OWL 2, respectively. In the context of OWL, the suffix (D) indicates the use of XML Schema[2] datatypes.

[2] `http://www.w3.org/TR/xmlschema11-2/`

Model-theoretic semantics

The semantics of description logics is model-theoretic and given by interpretations. An interpretation $\mathcal{I}$ is a tuple $\langle \triangle^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\triangle^{\mathcal{I}}$ is the interpretation domain, that is, a non-empty set of elements, and $\cdot^{\mathcal{I}}$ the interpretation function. The interpretation function maps concept names A in the knowledge base to sets $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$, role names to sets $r^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$, and individuals to elements in $\triangle^{\mathcal{I}}$. We call the set $C^{\mathcal{I}}$ the *extension* of the concept C in $\mathcal{I}$. The interpretation function for concepts, roles, and individuals in $\mathcal{ALC}$ is defined in Table 2.1.

Table 2.1: $\mathcal{ALC}$ constructors and semantics.

Constructor	Syntax	Semantics
Top concept	$\top$	$\triangle^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Concept negation	$\neg C$	$\triangle^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Concept intersection (conjunction)	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Concept union (disjunction)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential restriction	$\exists R.C$	$\{x \mid \exists y.^{\mathcal{I}} \langle x, y \rangle \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Universal restriction	$\forall R.C$	$\{x \mid \forall y.^{\mathcal{I}} \langle x, y \rangle \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$

An interpretation $\mathcal{I}$ is a *model* for an axiom α if it *satisfies* α; we write $\mathcal{I} \models \alpha$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in $\mathcal{I}$ for a subsumption $\alpha = C \sqsubseteq D$, and if $C^{\mathcal{I}} = D^{\mathcal{I}}$ in $\mathcal{I}$ for α an equivalence axiom $C \equiv D$. A model of a knowledge base $\mathcal{K}$ is an interpretation $\mathcal{I}$ in which *all* axioms in $\mathcal{K}$ are satisfied: $\mathcal{I} \models \mathcal{K}$. A *tautology* is an axiom which is always interpreted as $\top$; for example, the axiom $A \sqsubseteq A$ is a tautology, as $A^{\mathcal{I}} \subseteq A^{\mathcal{I}}$ holds in *all* models $\mathcal{I}$ of $\mathcal{K}$.

2.1.2 Standard reasoning services

The ability to convey information without having to explicitly state it is one of the main advantages of logic-based knowledge bases. A description logic *reasoner* is a piece of software which implements a decision procedure for the standard reasoning problems:

Consistency Given a knowledge base $\mathcal{K}$, determine whether there exists an interpretation $\mathcal{I}$ for $\mathcal{K}$ such that $\mathcal{I} \models \mathcal{K}$. If there exists such an interpretation, return 'true' (the KB is consistent), otherwise return 'false' (the KB

is inconsistent). All other reasoning services can be reduced to consistency checking in logics that support conjunction and negation.

Satisfiability A concept C is satisfiable if there is *some* model of $\mathcal{K}$ in which C is not empty: $C^{\mathcal{I}} \neq \emptyset$ in some $\mathcal{I}$ that is a model $\mathcal{K}$. C is unsatisfiable, denoted as $\mathcal{K} \models C \sqsubseteq \bot$, if it is mapped to the empty set in *all* models $\mathcal{I}$ of the knowledge base.

Subsumption One of the main reasoning task in DLs is *subsumption* between concept expressions, i.e. checking whether the extension of a (potentially complex) concept C in $\mathcal{K}$ is a subset of the extension of another concept D ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$). The subsumption between two concepts where C is necessarily interpreted as a subset of D is denoted as $\mathcal{K} \models C \sqsubseteq D$.

Equivalence Two concepts C and D are *equivalent* in the KB ($C \equiv D$), if it holds that all elements of $C^{\mathcal{I}}$ are in the set $D^{\mathcal{I}}$ and vice versa, i.e. if $C^{\mathcal{I}} = D^{\mathcal{I}}$, in all models $\mathcal{I}$ of $\mathcal{K}$. Equivalence is a special case of subsumption.

Instantiation Instantiation (or instance checking) checks which individuals are instances of a particular concept: given an individual a, a concept C, and a KB $\mathcal{K}$, return 'true' if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{K}$.

There exist a number of sound, complete, and terminating algorithms for providing the above reasoning services, which are implemented by description logic reasoners: *Tableaux* [DL96, Hor97, HST00, BS00] procedures attempt to build a finite tree-like model of the concepts in the knowledge base using *tableau rules*. The input is first translated into negation normal form (NNF) and then decomposed according to the respective tableau rules. The nodes and edges in the tree are labelled with the decomposed concepts and roles, respectively. A *clash* in the tree occurs when the algorithm attempts to label a node in the tree with contradictory concepts, such as an atomic concept A and its negation $\neg A$.

When no more rules are applicable to the concept fragments, or when a clash occurs, the algorithm terminates. If some clash-free tree (i.e. some model) can be found for a concept C, the concept is satisfiable. Likewise, if no clash-free tree can be found, the concept has no models, i.e. it is unsatisfiable. The types and usage of constructors in a description logic affect the types of rules used in a tableau algorithm, which in turn can increase the computational complexity of the reasoning algorithm. For instance, as some of the tableau rules are non-deterministic (i.e. involve choice), the algorithm has to perform backtracking if such a rule has been applied.

Entailments

An entailment η of a knowledge base $\mathcal{K}$ is an axiom that follows logically from $\mathcal{K}$. $\mathcal{K}$ entails η ($\mathcal{K} \models \eta$) if η is true in *all* models $\mathcal{I}$ of $\mathcal{K}$. The entailment relation is *monotonic* in DLs, which means that entailments are preserved when further axioms are added to the KB. Every axiom which is asserted in $\mathcal{K}$ is an entailment of the knowledge base, while there are other entailments which represent *implicit* knowledge in the KB. For example, the pet KB above explicitly asserts that $\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}$ and $\mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}$. From these statements we can conclude that every Cat is an Animal: $\mathcal{K} \models \mathsf{Cat} \sqsubseteq \mathsf{Animal}$.

Entailments are not restricted to any specific axiom type; for example, the axioms $\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}$ and $\mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}$ also entail the statement $\mathsf{Cat} \sqsubseteq \forall \mathsf{eats}.\mathsf{Animal}$. The set of all axioms which are valid in a description logic $\mathcal{L}$ and entailed by a KB $\mathcal{K}$ is called the *deductive closure* of $\mathcal{K}$:

Definition 2.1 (Deductive closure)**.** *The deductive closure $\mathcal{K}^*_\mathcal{L}$ of a knowledge base $\mathcal{K}$ is the set of $\mathcal{L}$-axioms entailed by $\mathcal{K}$, i.e. $\mathcal{K}^*_\mathcal{L} = \{\alpha \in \mathcal{L} \mid \mathcal{K} \models \alpha\}$. When clear from the context, the subscript $\mathcal{L}$ is dropped.*

Even KBs based on weakly expressive DLs have infinitely many entailments (i.e. the deductive closure is infinite). The simplest example is the *empty* knowledge base $\mathcal{K}$ over a non-empty signature $sig(\mathcal{K}) = \{A\}$ in the logic $\mathcal{ALC}$. The deductive closure of $\mathcal{K}$ then is the infinite set of all axioms that can be formed using A and the available constructors in $\mathcal{ALC}$:

$$\mathcal{K}^* = \{A \sqsubseteq A, A \sqcap A \sqsubseteq A, A \sqcup A \sqsubseteq A, A \sqsubseteq A \sqcap \top, \ldots\}$$

The focus when constructing and analysing a knowledge base often lies on the *concept hierarchy* of the KB, which is represented by its entailed atomic subsumption axioms. In the context of this thesis, we simply speak of an *entailment set* to denote a *finite* set of entailments of a knowledge base $\mathcal{K}$, restricted by some clearly specified criteria:

Definition 2.2 (Entailment set)**.** *An entailment set $\varepsilon_\mathcal{K}$ of a knowledge base $\mathcal{K}$ is a finite set of axioms $\{\alpha_i \mid i \in \{1..n\}\} \subseteq \mathcal{K}^*$.*

An in-depth discussion of the issue of specifying useful finite entailment sets follows in Chapter 3.

Incoherence

A concept C is *unsatisfiable* in a knowledge base $\mathcal{K}$ if there is no model $\mathcal{I}$ of $\mathcal{K}$ in which $C^{\mathcal{I}}$ is non-empty. This means that the concept cannot have any instances. Unsatisfiability is caused by contradictory statements, such as $C \sqsubseteq D \sqcap \neg D$. A KB that contains some unsatisfiable named concept is called *incoherent*.

Continuing with the previous example KB, we can induce incoherence in $\mathcal{K}$' $= \langle \mathcal{T}', \mathcal{A} \rangle$ by adding the last TBox axiom which causes the concept $\mathsf{SickCat}$ to be unsatisfiable:

Example 2.2.

$$\begin{aligned}\mathcal{T}' = \{&\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}, \mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal},\\ &\mathsf{Plant} \sqsubseteq \neg \mathsf{Animal}, \mathsf{Grass} \sqsubseteq \mathsf{Plant}\\ &\mathsf{PetOwner} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasPet}.\mathsf{Animal}, \exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal},\\ &\mathsf{SickCat} \equiv \mathsf{Cat} \sqcap \exists \mathsf{eats}.\mathsf{Grass}\}\\ \mathcal{A} = \{&\mathsf{Cat}(\mathsf{Molly}), \mathsf{Human}(\mathsf{Alice}), \mathsf{hasPet}(\mathsf{Alice}, \mathsf{Molly})\}\end{aligned}$$

The conflict is caused by the TBox axioms in $\mathcal{K}$' which entail that Cats are $\mathsf{Carnivores}$, thus eating only $\mathsf{Animals}$, but that instances of $\mathsf{SickCat}$ eat Grass, a concept which is a subconcept of Plant, thus known to be disjoint with Animal. As, according to our knowledge base, there cannot exist a $\mathsf{Carnivore}$ who also eats Grass, the concept $\mathsf{SickCat}$ cannot have any elements in any model $\mathcal{I}$ of $\mathcal{K}$': $SickCat^{\mathcal{I}} = \emptyset$. Therefore, the concept $\mathsf{SickCat}$ is said to be unsatisfiable, and the knowledge base $\mathcal{K}$' is incoherent.

Inconsistency

While incoherence itself does not cause any reasoner problems (as the concept simply corresponds to the empty set in all models), instantiation of unsatisfiable concepts causes the KB to be contradictory. If the axiom $\mathsf{SickCat}(\mathsf{Molly})$ is added to the KB in which $\mathsf{SickCat}$ is unsatisfiable, it is not possible for the KB to have any model in which this statement is satisfied. Such a knowledge base that has no models is called *inconsistent*, which is denoted as $\mathcal{K} \models \top \sqsubseteq \bot$.

An inconsistent KB has the property that it entails *everything* that can be expressed in the respective logic $\mathcal{L}$—since it is contradictory, it cannot possibly

make any meaningful statements about the knowledge it models. Different approaches to reasoning with inconsistent knowledge bases have been explored, such as *paraconsistent* reasoning [MHL07], or the selection of a consistent subset based on a relevance function [HvT05].

2.1.3 The Web Ontology Language OWL

From description logic to OWL

While 'ontology' is a term borrowed from philosophy, in computer science it describes a software artefact representing information about the entities in a domain, and the relationships between them [Gru93]. In the remainder of this thesis, we will use the term *ontology* (denoted by the letter $\mathcal{O}$) to refer to a description logic knowledge base which is represented in some machine-processable format, such as OWL.

OWL is a successor of the web ontology language DAML+OIL [Hor02], a description logic based ontology language with an RDF/XML syntax which evolved from merging DAML-ONT (a language developed by the DARPA Agent Markup Language programme) and OIL (Ontology Inference Layer, developed by the European *On-To-Knowledge* project). The first version of OWL, which is based on the expressive description logic $\mathcal{SHOIN}(D)$ and was described as a 'revision' of DAML+OIL, became an official W3C recommendation in February 2004.[3]

OWL may also be regarded as a more expressive successor to RDF (Resource Description Framework) [W3C04b], a language to describe relationships between entities using subject-predicate-object style *triples*, and RDFS (RDF Schema) [W3C04a], an extension of the RDF vocabulary which introduces 'meaningful' predicates such as `Class`, `subClassOf`, and `domain`.

OWL 2, the successor of OWL, was made a W3C recommendation in 2009 [W3C09]. It comprises two species of different expressivities, namely OWL 2 DL and OWL 2 Full. The underlying formalism of OWL 2 DL is the description logic $\mathcal{SROIQ}(D)$ [HKS06]. This highly expressive DL adds a range of constructors and axiom types to those of $\mathcal{ALC}$, such as complex role inclusions (represented by the letter $\mathcal{R}$), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), qualified number restrictions ($\mathcal{Q}$), and datatypes (D). While OWL 2 DL has the familiar description logic semantics (*Direct Semantics*) described above, OWL 2 Full [W3C12] has an RDF-based

[3] `http://www.w3.org/TR/owl-features/`

semantics, which is a superset of the OWL 2 Direct Semantics.

There exist several syntaxes for OWL [Hor10], such as the human-oriented Manchester Syntax [HDG+06], various RDF formats, and an OWL/XML serialisation. The axiom $\mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal}$ from the above example is written as follows in OWL Manchester Syntax (keywords are set in italics):

Class: Carnivore

SubClassOf: Animal *and* eats *only* Animal

In addition to the different syntaxes, OWL also introduces corresponding terms for the entities used in ontologies: *concepts* are referred to as *classes*, *roles* become *properties*, the top concept $\top$ is referred to as Thing, and the bottom concept $\bot$ as Nothing. For the remainder of this thesis, we will switch to using OWL lingo to refer to classes, properties, (sub-)expressions, and ontologies. Further, in the context of this thesis we will simply regard OWL 2 DL as a syntactic variant of $\mathcal{SROIQ}$, using the name OWL in place of OWL 2 DL, unless otherwise stated.

Usage of OWL

OWL was designed with the aim of providing an expressive machine-processable ontology language for use in various applications and domains. There exists a wide array of tools and libraries for creating and editing OWL ontologies, such as Protégé 4,[4] Top Braid Composer,[5] Swoop,[6] the OWL API,[7] a Java library that gives access to a large number of ontology editing tasks and reasoner interfaces, a number of highly optimised OWL reasoners, such as Pellet [SPC+07], HermiT [SMH08], and FaCT++ [TH06], as well as reasoners which are tuned towards specific subsets of OWL, such as *ELK* [KKS11], a reasoner for $\mathcal{EL}^{++}$ ontologies.

This broad set of OWL tools allows users to choose a suitable development environment for their respective application requirements, and improved tool support is being regarded as one of the main reasons for the growing use of OWL ontologies as a knowledge representation mechanism [HPS08].

[4] `http://protege.stanford.edu/`
[5] `http://www.topquadrant.com/products/TB_Composer.html`
[6] `https://code.google.com/p/swoop/`
[7] `http://owlapi.sourceforge.net/`

OWL 2 profiles There exist three named 'profiles' for OWL 2, syntactic subsets of OWL 2 DL that are tailored towards different applications, which trade expressivity of the language for efficient reasoning:

The OWL 2 EL profile is a tractable fragment of OWL 2 which is based on the description logic $\mathcal{EL}^{++}$ [BBL05]. OWL 2 EL omits some of the more expressive OWL 2 constructors in favour of efficient reasoning (polynomial time), which makes it attractive for use in ontologies that 'contain very large numbers of properties and/or classes' [owl09].

OWL 2 QL (Query Language), which is based on the *DL-Lite* family of description logics [ACKZ09], has been defined for use in applications which focus on query answering over large amounts of instance data. The queries executed against OWL 2 QL ontologies can be rewritten into SQL queries, which allows storing instance data in a standard relational database, thus significantly improving query performance. OWL 2 QL is used for ontologies in Ontology-Based Data Access (OBDA) systems, which combine OWL 2 QL reasoning with efficient querying over relational databases [CGL$^+$11, RMC12].

Reasoning systems for ontologies in the OWL 2 RL (Rule Language) profile can be implemented using rule-based reasoning engines. Similar to OWL 2 QL, the profile restricts the use of constructors to certain positions in axioms, e.g. universal restrictions and negation are only allowed on the RHS of axioms. While OWL 2 RL allows the use of a wider range of expressive constructors than the other two OWL 2 profiles, these positional restrictions lead to efficient reasoning performance.

Semantic Web The idea of a Semantic Web [BLHL01] marks a move away from a human-oriented 'web of documents' towards a machine-oriented 'web of data'. By enriching web pages with semantic content we can express relationships between entities on the web, thus allowing understanding of those relationships by automated processes rather than direct human processing. This 'meaningful' interlinking of web content is thought to improve searching for and integrating information on the web.

As a 'source of shared, precisely defined terms' [Hor02], ontologies play a key role in the Semantic Web, with the majority of ontologies found on the web using OWL or RDF(S) [Car07]. Examples of prominent OWL ontologies

on the web include the BBC Programmes and Wildlife ontologies,[8] and the set of geographical ontologies created by Ordnance Survey.[9] However, while OWL has been designed specifically as an ontology language for the Semantic Web, its *perceived* complexity and unpredictable reasoning performance still prohibit wider uptake by web application developers, for whom 'a little semantics' [Hen07] in the form of RDF may often seem sufficient.

Medical informatics and life sciences OWL ontologies are frequently used for the encoding of biological and medical knowledge as part of larger information systems. Examples for intensively used and maintained medical OWL ontologies include the SNOMED CT ontology[10] which contains medical terminology used in electronic health records, the OWL version of the International Classification of Diseases (ICD-10) catalogue[11] released by the World Health Organization (WHO), and the National Cancer Institute (NCI) thesaurus[12] which covers knowledge related to cancer, such as anatomy, findings, drugs, and genes.

There exists a vast array of biological ontologies of varying size and expressivity, such as the Gene Ontology, which aims to 'standardize the representation of gene and gene product attributes', the Experimental Factor Ontology (EFO) which models variables in biological experiments, and various ontologies describing the anatomy of species. Finally, the NCBO BioPortal [NSW+09] is a prominent curated collection of biomedical ontologies that were created by a number of research groups which has received some attention in recent years as a popular test corpus for OWL ontology research [BHPS11, HPS11, HPS12, MMIS12].

2.2 Errors in OWL ontologies

OWL is a highly expressive ontology language which offers users a wide range of constructors for modelling domain knowledge at a high level of precision. The downside of this expressivity is that constructors can be misinterpreted by users, and combinations of otherwise correct axioms may lead to undesired side-effects. Beyond side-effects caused by human users, common engineering tasks such as

[8] `http://www.bbc.co.uk/ontologies/`
[9] `http://www.ordnancesurvey.co.uk/oswebsite/ontology/`
[10] `http://www.ihtsdo.org/snomed-ct/`
[11] `http://www.who.int/classifications/icd/en/`
[12] `http://ncit.nci.nih.gov/`

translation of an ontology into OWL from some other formalism, automated generation of an ontology using some input source, or integration of an existing ontology, may also introduce errors into an ontology.

We mainly distinguish between two different types of errors: *logical errors*, which can be detected through the use of a reasoner, and *non-logical errors*, which range over incorrect statements with respect to the domain knowledge modelled in the ontology, modelling inconsistency, reasoner performance issues, and annotation problems. In this section, we will give an overview of the different errors users may encounter in the OWL ontology development process, and describe how these errors are commonly detected and repaired.

2.2.1 Logical errors

Unsatisfiable and tautological classes

A common logical error is the unsatisfiability of a named class A in an ontology $\mathcal{O}$, which implies that A cannot have any instances. While the unsatisfiability of a class *may* be intentional (e.g. for testing purposes as suggested by the Protégé 4 OWL tutorial [Hor11b], to explicitly prohibit a certain definition for a class, or to introduce a new name for $\bot$), it mostly indicates a modelling error caused by contradictory statements. Unsatisfiable classes are easily detected by a reasoner and are commonly highlighted as errors in OWL editing tools, for example by rendering them in red colour, or by arranging them as a subclass of $\bot$.

Likewise, we call a class B in an ontology $\mathcal{O}$ *tautological* if $\mathcal{O}$ entails that $\top \equiv B$, that is, every class in the domain is entailed to be a subclass of B, and every individual is entailed to be an instance of B. As with unsatisfiable classes, a tautological class may be introduced on purpose to achieve that very same effect. However, we know that tautological classes often occur accidentally and without ontology users noticing or understanding why something is entailed to be equivalent to $\top$ [HPS09]. This phenomenon may also cause unwanted side effects, such as the unsatisfiability of a class that is asserted to be disjoint with the tautological class, as we have seen in the *Movie* ontology example [HPS09], which we will describe in Section 2.3.3. While tautological classes can also be detected by a reasoner, they are generally not treated as errors in OWL tools.

Inconsistency

By instantiating an unsatisfiable class, i.e. specifying that an individual is an instance of an unsatisfiable class, an ontology becomes *inconsistent*, or *contradictory*. Inconsistency can also be caused by a TBox which contains axioms that prohibit any class from having a non-empty extension, for example by using an axiom of the type $A \sqcup \neg A \sqsubseteq B \sqcap \neg B$. Inconsistency is a severe logical error as it causes the ontology to entail *every* axiom, which makes it impossible (under standard semantics) to infer meaningful information from the ontology; this renders the ontology useless. In order to prevent the inconsistency of an ontology, it is highly desirable to *repair* any unsatisfiable classes as soon as they arise in the ontology engineering process.

Wrong entailments

Some logical errors, such as incoherence and inconsistency, can be detected using a reasoner, as detection is simply reduced to entailment. Factual errors (wrong and unwanted entailments), however, can only be spotted by a user with the appropriate domain knowledge.

While there may be some knowledge in the world which we can consider facts (such as 'the parent of every human is a human'—for the time being, there is little to argue about this statement), we often come across knowledge and relationships which are and *cannot* be clearly defined. Thus, we can say that every ontology is subject to an *ontological commitment* which prescribes relations in the ontology depending on the *view* we have on a domain. Going back to the parent–human relationship, what happens if we build a knowledge base that is a representation of creatures from Greek mythology? In such an ontology we can imagine the need to allow a human to have a non-human parent; yet, given our restriction that humans have only human parents, this would be impossible.

While mythological creatures may be a very specific example, we can quickly see cases in which such a restriction can lead to more fundamental debates. A frequently occurring issue is that of gender: general knowledge tells us that there are two genders, and every person *has* to be either male or female (but not both), which we may model as such in an ontology. And yet, this is clearly an issue which ranges far beyond knowledge modelling techniques, as the gender binary has long been the focus of fundamental biological and political debate. When talking

about *errors* in an ontology, we have to bear in mind that some errors may not be statements which are intrinsically wrong, but simply *wrong in the context of the ontology.*

Besides factual errors caused by ambiguities and the difficulty of modelling certain aspects of a domain, wrong entailments can also stem from incorrect use of OWL constructors. We have a good understanding of the common errors that users make when constructing OWL ontologies [RDH+04, KPSC06, RCVB09]. Some of these errors include confusing 'and' and 'or', subsumptions and equivalences, 'forgetting' previously defined property restrictions, and introducing axioms which do not have any effect on the model of the ontology. Given a catalogue of such *antipatterns*, an OWL tool can point out axioms that are potentially erroneous and provide further explanation of the semantics to a user. Kalyanpur et al. [KPSC06] use a set of common modelling errors in a heuristics-based repair tool in the Swoop ontology editor which suggests removal or modification of axioms based on, amongst other aspects, the likelihood of them being incorrect.

Non-entailments

Missing entailments, or non-entailments, are frequently occurring errors in OWL ontologies which are particularly hard to detect. When constructing an ontology, developers expect certain entailments, such as an obvious subsumption between two classes, to be caused by the axioms they add; if an expected (and desired) entailment does not follow from the ontology, this may be considered an error. Roussey et al. [RCVB09] found that ontology developers frequently add axioms to ontology that either do not have the desired effect, or no effect at all. In these cases, *ontology diff* tools (e.g. [FMV10, GPS11b, KŠK11]) can help users spot whether any modifications to an ontology had the desired effect; beyond the detection of ineffectual modifications, however, dealing with non-entailments is a non-trivial task.

A structured approach to eliciting missing entailments and adding them to the ontology is *ontology completion* [BGSS07], which is based on methods from the area of Formal Concept Analysis (FCA) [GSW05]. An ontology completion tool presents users with a series of potential entailments, e.g. a subsumption between two classes, asking them to accept or reject the presented entailment. In case of an accepted entailment, an axiom is added to the ontology to explicitly state the relationship; otherwise, the tool asks the user to produce a *counter-example* to

ensure that the axiom is not entailed.

If we look at non-entailments from a different angle, the non-entailment of a *negative* entailment may also be considered an error. OWL is based on the *Open World Assumption* which means that anything that is not explicitly forbidden is considered to be a 'don't know' rather than a 'no'. For instance, in the above example ontology we did not explicitly state that Human and Cat are disjoint classes; that is, at a later point, we could add the statement Human(Molly) to the existing Cat(Molly) without causing a logical error in the ontology. While the error then falls into the category of unwanted entailments, it may be useful to *prevent* such problems before they even occur. Similar to ontology completion, *semantic clarification* [Sch05a] aims to prevent unwanted entailments by strengthening the ontology with disjointness axioms where appropriate.

2.2.2 Non-logical errors

Structural irregularities

There exists a number of ontology design patterns [Gan05, AAKS08] which provide solutions for situations that are difficult to model in OWL, such as representing n-ary relations. But even without such formal patterns, ontologies often exhibit syntactic regularities due to habits or training of ontology engineers [MMIS12]. Such regularities include 'good practice' strategies such as adding domain and range axioms for every newly introduced object property. When a specific design pattern or design style is chosen for an ontology, it should be adhered to across the ontology in order to maintain a uniform modelling style. Therefore, deviating from a given structure and existing regularities may be considered an error in the ontology. Ontology pattern inspection tools, such as the RIO Protégé 4 plugin [MIS12], can assist ontology developers in detecting regularities and deviations in OWL ontologies.

Annotation errors

Annotations in OWL ontologies are frequently used to convey information about an entity and its relations beyond the logical content of the ontology. One such example is the NCI Thesaurus, in which on average (across a set of 86 subsequent versions of the ontology) over 84% of the axioms are annotations [GPS11a]. It is possible for an annotation to be erroneous, for example, by containing false or

missing information about the entity it annotates. Other than manual inspection (or specifically tailored text processing of the annotation strings), there is no way to detect an incorrect annotation.

Performance problems

The performance of reasoners for tasks such as classification and query answering over OWL ontologies is one of the main focus points of OWL research. As with 'regular' software, the general user acceptance and usefulness in a production environment of OWL depends, amongst other reasons, on its efficiency.

However, OWL ontologies often turn out to be rather unstable in terms of their performance; even minor changes can seemingly randomly lead from short classification times in the range of seconds to impractical times. Tools which detect *performance hot spots* [LS08, GPS12b] can assist ontology developers in identifying ontology subsets which cause a steep increase in classification times. Based upon the hot spot information provided by such a tool, the ontology developer can then remove or rewrite the hotspot-axioms in order to improve the reasoner performance.

2.2.3 Debugging ontologies

In the context of OWL ontology engineering, the debugging stage involves the process of *detecting* an error in an ontology (e.g. an unsatisfiable class, or an incorrect entailment), finding the *source* of the error, i.e. the information stated in the ontology which causes the error, and finally, *repairing* the ontology by modifying or removing (some of) the problematic information. Losing as little *correct* information as possible in the repair stage is a key aspect of successful debugging.

In the case of logical errors, the repair process can be performed manually, using a reasoner as a tool to classify the ontology, then searching the class hierarchy for unwanted entailments (potentially aided by a suitable visualisation in an ontology editor, such as Protégé 4 which arranges all unsatisfiable classes under the class Nothing), tracing the source of the error by inspecting related statements in the ontology, and then modifying those statements that are considered to be incorrect.

Due to the potential size and complexity of OWL ontologies, this manual approach quickly leads to a tedious process of searching through the entire ontology. Moreover, it may lead to non-optimal repairs, where more information than required is removed or modified for the purpose of fixing the entailment. Anecdotes of ontology developers 'ripping out' parts of the ontology in order to remove an entailment show that this approach is far from ideal and indicates a clear need for additional debugging support.

One of the first approaches to explaining subsumption in description logic ontologies was introduced by McGuinness [McG96] and Borgida [MB95], using proof-style explanations for entailments of the CLASSIC knowledge representation system [PSMB+91]. This proof-based approach was later extended by Borgida [BFH99] to ontologies in the description logic $\mathcal{ALC}$. One of the earliest implementations of debugging facilities in an OWL ontology editor is the explanation component *MEX* in the OntoTrack editor [LN04, LvHN05]. *MEX* presents natural language proof-style explanations which are based on a tableaux trace as proposed by Borgida et al. [BFH99] and later extended from $\mathcal{ALC}$ to cover nearly the complete OWL Lite sub-language of OWL 1 [LvHN05]. In line with these proof-based approaches, Deng et al. proposed a resolution based explanation framework [DHS05] which was restricted to unsatisfiability and inconsistency of $\mathcal{ALC}$ ontologies.

However, it was not until the introduction of *MUPS* (Minimal Unsatisfiability Preserving Sub-TBoxes) [SC03], later named *justifications*, that explanation support for description logic ontologies became a key aspect of OWL ontology research, spawning a substantial body of work and several ontology debugging tools.

2.3 Justifications for entailments of ontologies

Justifications are the most prominent form of debugging support for entailments of OWL ontologies. They are based on a work by Schlobach and Cornet [SC03, SHCvH07], who developed a strategy for *pinpointing* the causes of unsatisfiable classes in the medical ontology *DICE*. A justification $\mathcal{J}$ for an entailment η of an ontology $\mathcal{O}$ is a minimal subset of $\mathcal{O}$ which is sufficient for η to hold:

Definition 2.3 (Justification)**.** *$\mathcal{J}$ is a justification for $\mathcal{O} \models \eta$ if $\mathcal{J} \subseteq \mathcal{O}, \mathcal{J} \models \eta$ and, for all $\mathcal{J}' \subset \mathcal{J}$, it holds that $\mathcal{J}' \not\models \eta$.*

A justification $(\mathcal{J}, \eta)$ is defined with respect to a single entailment η; in order to describe the set of *all* justifications for a single entailment, or for the justifications for all entailments in an entailment set $\varepsilon_{\mathcal{O}}$, we introduce the notion of *justification sets*:

Definition 2.4 (Justification set)**.** *Given an ontology $\mathcal{O}$ and an entailment η, the justification set $Justs(\eta)$ is the set of all justifications $\{\mathcal{J}_1 \dots \mathcal{J}_n\}$, $\mathcal{J}_i \subseteq \mathcal{O}$, for η. Likewise, the justification set $Justs(\varepsilon_{\mathcal{O}})$ of an entailment set $\varepsilon_{\mathcal{O}}$ is the set of all justifications $\mathcal{J}_i$ for all $\eta_k \in \varepsilon_{\mathcal{O}}$.*

While Schlobach and Cornet focused on *MUPS* and Minimal Incoherence-Preserving Sub-TBoxes (*MIPS*), that is, minimal entailing subsets for *some* cause of incoherence in $\mathcal{O}$) in unfoldable $\mathcal{ALC}$ TBoxes, the concept of explanations based on minimal entailing subsets is applicable to arbitrary entailments. Kalyanpur et al. [KPSH05] extended the idea of *MUPS* to unsatisfiable classes in general OWL ontologies, presenting users with the *sets of support* (which were later named *justifications*, a term borrowed from belief revision [Gär88, Neb90]) for an entailment. Beyond its application for OWL ontology debugging, computing minimal unsatisfiable cores is also a technique for generating proofs in the SAT community [LMS04].

The minimality of $\mathcal{J}$ implies that removing any one of its axioms *breaks* the entailment η. For any entailment η of $\mathcal{O}$, η itself is a justification if η is asserted in $\mathcal{O}$, and there can be multiple justifications (exponentially many [BPS07a]) for a single entailment.

Recall our example from the animal domain which entails that the class SickCat is unsatisfiable:

Example 2.3.

$$\begin{aligned}\mathcal{T}' = \{&\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}, \mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal},\\ &\mathsf{Plant} \sqsubseteq \neg\mathsf{Animal}, \mathsf{Grass} \sqsubseteq \mathsf{Plant}\\ &\mathsf{PetOwner} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasPet}.\mathsf{Animal}, \exists \mathsf{eats}.\top \sqsubseteq \mathsf{Animal},\\ &\mathsf{SickCat} \equiv \mathsf{Cat} \sqcap \exists \mathsf{eats}.\mathsf{Grass}\}\\ \mathcal{A} = \{&\mathsf{Cat}(\mathsf{Molly}), \mathsf{Human}(\mathsf{Alice}), \mathsf{hasPet}(\mathsf{Alice}, \mathsf{Molly})\}\end{aligned}$$

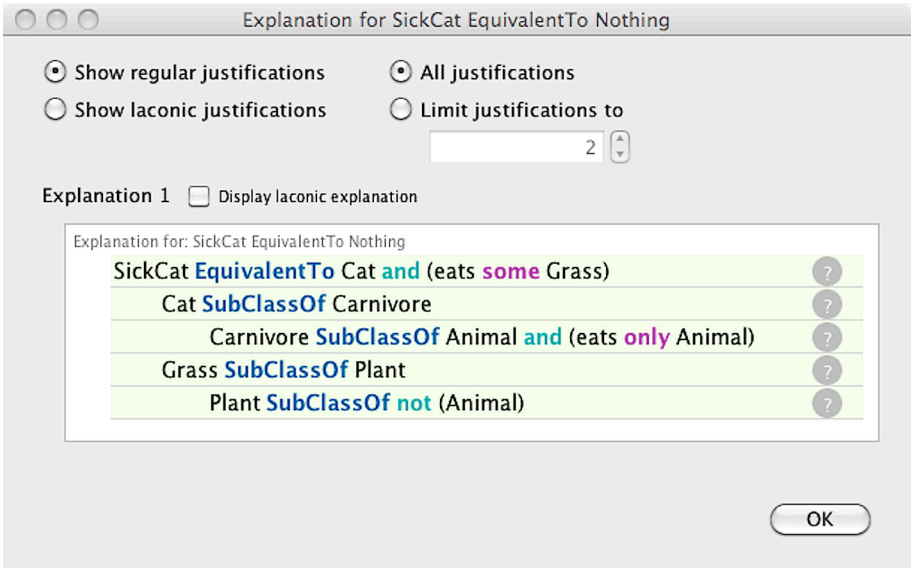


Figure 2.1: A screenshot of the *Explanation* tab in Protégé 4.

The following axiom set is a justification for $\mathcal{K}' \models \mathsf{SickCat} \sqsubseteq \bot$:

$$\begin{aligned}\mathcal{J} = \{&\mathsf{Cat} \sqsubseteq \mathsf{Carnivore}, \mathsf{Carnivore} \sqsubseteq \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Animal},\\ &\mathsf{SickCat} \equiv \mathsf{Cat} \sqcap \exists \mathsf{eats}.\mathsf{Grass}, \mathsf{Grass} \sqsubseteq \mathsf{Plant},\\ &\mathsf{Plant} \sqsubseteq \neg \mathsf{Animal}\} \models \mathsf{SickCat} \sqsubseteq \bot\end{aligned}$$

In addition to limiting the number of axioms a user has to analyse when presented with a justification, OWL ontology editors generally *order* and *indent* the axioms in justifications, which significantly increases readability. Figure 2.1 shows the above justification as displayed in Protégé 4 with ordered and indented axioms.

The concept of justifications for description logic ontologies has been widely studied over the past decade, with particular focus on developing efficient algorithms for the computations of all justifications for a given entailment, e.g. [Sch05b, Sun08, DQJ09]. There also exist a number of debugging approaches that are closely related to justifications, such as the computation of maximally satisfiable terminologies (MSSs) [MLBP06, MLP06], which is based on a tableau-like procedure, and the *pinpointing formula* approach proposed by Baader and Peñaloza [BPS07b, BP08b, JQH08, PS10].

Rather than directly computing the set of all justifications for an entailment, a pinpointing formula is a compact representation of the set of all justifications for an entailment [BPS07b, BP08b, PS10]. A pinpointing formula is a monotone

Boolean formula where each propositional variable represents an axiom in the ontology. The set of justifications for an entailment can then be derived from the pinpointing formula as it corresponds to all valuations of the formula. An example of a pinpointing formula is given in [BPS07b]:

Example 2.4.

$$\begin{aligned} (\alpha_1) \quad & A \sqsubseteq \exists r.A \\ (\alpha_2) \quad & A \sqsubseteq Y \\ (\alpha_2) \quad & \exists r.Y \sqsubseteq B \\ (\alpha_4) \quad & Y \sqsubseteq B \end{aligned}$$

The four axioms labelled with α_1 to α_4 entail $A \sqsubseteq B$. We can find two justifications for this entailment: $\mathcal{J}_1 = \{\alpha_2, \alpha_4\}$ and $\mathcal{J}_2 = \{\alpha_1, \alpha_2, \alpha_3\}$. If we treat the axiom labels as propositional variables, the two justifications correspond to the 'true' valuations of the pinpointing formula $\alpha_2 \wedge (\alpha_4 \vee (\alpha_1 \wedge \alpha_3))$.

2.3.1 Computing justifications

There exist various approaches to directly computing *one* or *all* justifications for a given entailment, with 'find all' algorithms generally using a 'find one' algorithm as a subroutine. We categorise these algorithms into *glass-box* and *black-box* approaches [KPSH05]: glass-box techniques generally rely on information provided by a DL reasoner, which requires modification of reasoner internals in order to utilise it for justification computation. By contrast, black-box techniques only use a reasoner as an *oracle* to perform entailment checks.

Glass-box techniques

Glass-box justification finding techniques generally rely on the modification of a description logic reasoner to keep track of the axioms required for an entailment to hold, such as those axioms causing a *clash* in a tableaux procedure. More precisely, as the problem of explaining arbitrary entailments, e.g. subsumption between classes, can be reduced to a consistency check, this technique can be applied to find justifications for both unsatisfiable classes as well as arbitrary entailments. The tracking strategy applied in glass-box approaches using

tableaux reasoners is known as *tracing* [KPSH05], which is based on an algorithm first proposed by Baader and Hollunder [BH95]. Beyond tableaux-based reasoners, approaches extending the $\mathcal{EL}^{++}$ subsumption algorithm [BPS07a] and automata-based approaches [BP08a, BP10] also fall under the label of 'glass box' justification generation. A fair number of approaches to explaining entailments of description logic KBs uses glass-box techniques to present users with reasoner traces in the form of axiom sets, natural language, or some form of visualisation [SC03, LH05, Kwo05, MLBP06].

Black-box techniques

Expand-contract approach A simple technique for finding *one* justification for an entailment η of an ontology $\mathcal{O}$ is the expand-contract approach [KPHS07]: in the *expansion phase*, axioms from the given ontology $\mathcal{O}$ are incrementally added to an empty ontology $\mathcal{O}'$, performing an entailment check after each addition until it is found that $\mathcal{O}'$ entails η. In order to generate a minimal justification from $\mathcal{O}'$, the *contraction phase* then removes superfluous axioms from $\mathcal{O}'$, performing an entailment check after each removal. This approach is known as *black-box* technique, as it only requires an out-of-the-box reasoner to perform the entailment checks. Optimisations of this algorithm focus on reducing the number of expensive entailment checks by employing a *divide-and-conquer* [FS05, SFJ08] or a *sliding-window* technique [Kal06] in the contraction phase.

Hitting set tree algorithm In order to find not only one but *all* justifications for an entailment, Schlobach [Sch05b] first proposed the use of Reiter's hitting set tree (HST) algorithm [Rei87, GSW89] for the computation of *MUPS* and *MIPS*. The algorithm originates from the field of *model based diagnosis*, which describes the process of finding diagnoses for faults in a system comprised of components. A minimal conflict set is a minimal set of such components which causes a system fault, whereby a diagnosis is a minimal *hitting set* across the set of minimal conflict sets. Given a set C of minimal conflict sets, a hitting set for C is a set $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \emptyset$ for each $S \in C$. Minimal conflict sets and diagnoses correspond to our notion of justifications and minimal repairs, respectively.

Reiter's algorithm constructs a hitting set tree in order to find all minimal hitting sets over a given set of minimal conflict sets in a diagnosis problem. The

nodes in the HST are labelled with minimal conflict sets, and the edges are labelled with components (axioms). For the purpose of finding justifications, the HST algorithm is initialised by computing a single justification using any glass-box or black-box *find one* algorithm, which suffices to generate the tree for *all* justifications, as shown in [KPHS07].

Optimisations, such as *early path termination* and *justification reuse* [KPHS07] help ensure that the algorithm terminates in practical time. Horridge [Hor11a] showed that it is possible to compute all justifications for direct atomic subsumptions from over 90% of the ontologies in a diverse test corpus; for the remaining ontologies, however, the algorithm did not terminate in the given time due to the large size of the constructed HST, or due to a timeout on entailment checks.

Modularisation

A *module* $\mathcal{M}$ of an ontology $\mathcal{O}$ is a subset of $\mathcal{O}$ which contains all axioms that are relevant for some seed signature $\Sigma \subseteq sig(\mathcal{O})$. There exist various types of modules and algorithms for efficient computation of a module for a given seed signature, see, for example [CPSK06, CHKS07, SSZ09]. It has been shown that *syntactic locality based modules* are *depleting*, which means that a module generated based on the signature of an entailment contains all justifications for this entailment [CHKS07]. Further, for any given seed signature, there exists a *unique* and *minimal* locality based module [CHKS08].

When used in the justification computation process, a syntactic locality based module $\mathcal{M}$ is computed for the signature of an entailment, and the justification computation deals only with $\mathcal{M}$ rather than the full ontology $\mathcal{O}$ [Sun08, DQJ09]. Typically, the number of axioms in a module is small compared to the whole ontology [Sun08], which leads to reduced computational load both in the expansion and contract phases, as well as for entailment checks. A study by Suntisrivaporn et al. [SQJH08] using randomly selected entailments from three fairly large and complex OWL ontologies (Galen, NCI, and Gene Ontology) found that the average size of syntactic locality based modules was only between 0.05% and 1.6% of the whole ontology. Consequently, the justification computation performance was drastically improved by around two orders of magnitude, making it possible to compute justifications even for entailments on which the algorithm using the full ontology timed out. As the overhead for module computation is neglectable compared to the overall justification computation time, and due to the significant

improvements obtained through modularisation, it is now considered a standard optimisation used in justification computation approaches for OWL ontologies.

Performance

Generally, glass-box techniques for finding single justifications are considered to be more efficient than black-box techniques, as the justifications are generated almost 'automatically' as a by-product of the classification process. And indeed, in an extensive analysis of various justification computation algorithms, Horridge [Hor11a] found that using a glass-box *find one* algorithm as subroutine can improve performance by an order of magnitude for some ontologies. However, given the heavy modifications required to integrate tracing with a reasoner, and the fact that currently only the Pellet reasoner supports tracing, we may consider black-box techniques as a more accessible alternative to glass-box approaches.

Regarding the feasibility of computing justifications for OWL ontologies used in practice, Horridge [Hor11a] found that it is highly likely that we can find *all* justifications for an entailment in reasonable time. His study of 72 ontologies from the NCBO BioPortal showed that for the majority of ontologies in the set (all but 7), all justifications for 99% of the entailed direct atomic subsumptions could be computed in less than 10 seconds.

2.3.2 Justification-based repair

While justifications offer ontology users a focused view on the subset(s) of the ontology which are relevant to an entailment, they do not provide much guidance as to which steps a user has to take in order to repair the erroneous entailment. Rather, the user is expected to analyse the axioms in the justification and identify a set of axioms $\mathcal{R}$ which can be either removed or modified in order to repair the entailment. A repair over the set of justifications for an entailment η is defined as follows [Hor11a]:

Definition 2.5 (Repair)**.** *Given $\mathcal{O} \models \eta$, the set of axioms $\mathcal{R}$ is a* repair for η *in $\mathcal{O}$ if $\mathcal{R} \subset \mathcal{O}$, $\mathcal{O} \setminus \mathcal{R} \not\models \eta$, and there is no $\mathcal{R}' \subset \mathcal{R}$ such that $\mathcal{O} \setminus \mathcal{R}' \not\models \eta$.*

More informally, a repair is a *hitting set* $\mathcal{R}$ over the set of justifications for η in $\mathcal{O}$, i.e. for each justification $\mathcal{J}$ for η in $\mathcal{O}$ there is at least one axiom in $\mathcal{J}$ which is contained in $\mathcal{R}$. As there are multiple possible repairs for a given set of justifications, we may want to find a *cardinality-minimal* repair, that is, a repair

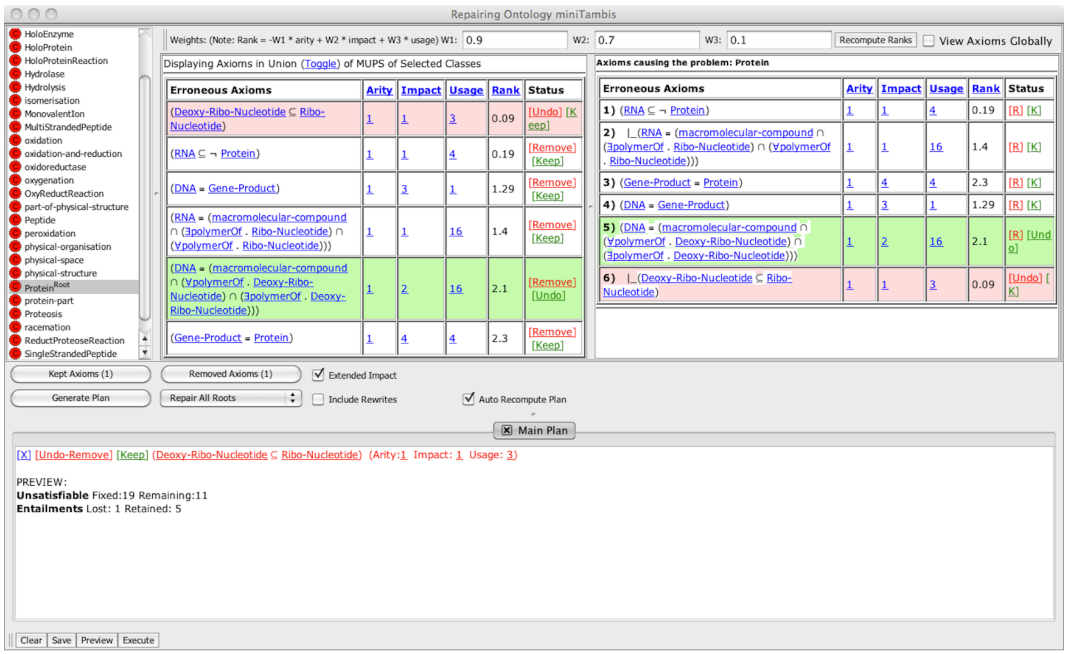


Figure 2.2: A screenshot of the *Repair* tool in Swoop.

$\mathcal{R}$ such that there is no repair $\mathcal{R}$' with fewer axioms than $\mathcal{R}$, as this implies removing or modifying the smallest possible number of axioms.

The quality of a repair does not only depend on the number of axioms to be removed or modified, but also on the amount of useful information in the ontology which is *lost* through such modifications. Thus, a suitable repair for an entailment may not be cardinality-minimal, but rather dependent on the *frequency* (*power*, *arity*) and *impact* [KPSC06] of the axioms in the justifications.

The *impact* [KPSC06] of an axiom in a justification is the number of entailments (from some clearly defined finite entailment set, such as the set of entailed atomic subsumptions) that are lost when removing the axiom from the ontology. Given a set of justifications for an entailment (or set of entailments), the *frequency* of an axiom is the number of justifications it occurs in, that is, the number of justifications that can be *broken* by removing the axiom.

Kalyanpur et al. [KPSC06] approached the problem of finding a suitable repair for unsatisfiable classes by introducing a *ranking* on the axioms in justifications. Axioms are ranked according to their frequency, their impact, manually specified test cases by a user (which is an extension to the default impact in order to ensure that specific entailments are preserved), provenance information about the axiom (author, source reliability, time added or modified) and usage of the

terms in the axiom signature across the ontology. The repair tool in the Swoop [KPS+06] editor allows users to specify the weights of the various ranking features, then recommends the preservation or removal of high- or low-ranked axioms, respectively. A screenshot of Swoop's debugging panel is shown in Figure 2.2. While this kind of elaborate debugging support sounds promising, there have not been any in-depth user studies to confirm whether and to which extent ontology developers use the tool and how they benefit from it.

2.3.3 Understanding individual justifications

While the main focus of justification research has been on the performance of justification finding algorithms, in recent years the issue of *understanding* justifications has been receiving increased attention. We know that justifications can significantly reduce user effort by allowing them to focus on a small, *relevant* subset of an ontology; and yet, justifications do not always help all users in understanding the actual *reason* why an entailment holds and in finding a suitable repair.

Fine-grained justifications

By definition, a justification is a minimal entailing subset of an ontology; that is, a justification contains axioms as they are *asserted* in the ontology. This means that the axioms in a justification can contain *superfluous* parts, i.e. subexpressions on the LHS or RHS of the axiom which do not contribute to the entailment. While such superfluous parts can distract users from the actual cause of an entailment, thus making the justifications more difficult to understand, removing axioms with superfluous parts in the repair process also means a loss of possibly valuable information.

In order to cope with the issue of superfluity, the notion of *fine-grained* justification was first introduced by Kalyanpur et al. [KPC06]. This approach was based on the idea of re-writing justification axioms in a normalized form, then splitting the axioms across intersections and only preserving those axioms which are required for the entailment in question to hold. The strategy was implemented in the Swoop editor, using strike-out and colour highlighting to indicate superfluous expressions in axioms—a simple, yet visually effective technique. Lam et al. [LPSV06] directly integrated the idea of fine-grained justifications into a repair

tool, computing the impact of each axiom rewriting based on the modification of its *subexpressions*.

Based on these first attempts, Horridge et al. [HPS08] firstly proposed a definition for *laconic* justifications: a laconic justification is a justification which does not contain any superfluous parts, with every subexpression being as *weak* as possible [HPS08]. This implies that every subexpression in a laconic justification is relevant to the entailment. The authors [HPS08] also describe a method to compute the *preferred* laconic versions of a justification which results in a unique correspondence between a justification and its laconic variants. In short, the process

- removes any subexpressions from axioms which are not relevant for the entailment to hold.
- derives a single subsumption axiom from an equivalence where possible.
- substitutes class names with the $\top$ where possible.
- weakens number restrictions to the smallest number possible.

The following example illustrates several aspects of non-laconicity in a justification:

Example 2.5.

$$\mathcal{J} = \{A \sqsubseteq B \sqcap \leq 2r.C, \exists r.C \sqsubseteq D\} \models A \sqsubseteq D$$

First, we can reduce the intersection in the first axiom to the single expression $\leq 2r.C$ without affecting the entailment. Second, this expression can then be weakened to $\leq 1r.C$. Third, the class name C in both axioms can be substituted with $\top$. This results in the laconic justification $\{A \sqsubseteq \leq 1r.\top, \exists r.\top \sqsubseteq D\}$.

A survey [Hor11a] of 72 of OWL ontologies from the NCBO BioPortal[13] revealed that non-laconic justifications are prevalent across OWL ontologies; indeed, the majority of the surveyed ontologies (69 out of 72) contained at least some non-laconic justification for an atomic subsumption, with 29 ontologies containing over 50% non-laconic justifications.

Masking

Justification *masking* [HPS08, HPS10a] occurs when the actual number of *reasons* why an entailment holds differs from the number of justifications that can be

[13] `http://bioportal.bioontology.org/`

found for the entailment. There exist different types of masking, which are mainly caused by axioms containing superfluous expressions, i.e. being non-laconic. The examples given in [HPS10a] focus on unsatisfiable classes; it can easily be shown, however, that masking also occurs for arbitrary entailments.

Internal masking describes a situation where the justification contains more than one reason why an entailment holds, which means that there exist multiple laconic versions of the justification.

Example 2.6.

$$\mathcal{J} = \{A \sqsubseteq B \sqcap C, B \sqcup C \sqsubseteq D\} \models A \sqsubseteq D$$

Example 2.6 shows a justification $\mathcal{J}$ for $A \sqsubseteq D$ which contains two reasons for the entailment. The laconic versions of $\mathcal{J}$ are:

$$\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq D\}$$
$$\mathcal{J}_2 = \{A \sqsubseteq C, C \sqsubseteq D\}$$

External masking involves other axioms from the ontology which do *not* occur in any justifications for an entailment.

Example 2.7.

$$\mathcal{O} = \{A \sqsubseteq C \sqcap D, C \sqsubseteq D\} \models A \sqsubseteq D$$

In the ontology shown in Example 2.7 there exists only one justification for the entailment $A \sqsubseteq D$, namely the first axiom $A \sqsubseteq B \sqcap C$. It is clear to see, however, that the two axioms in $\mathcal{O}$ combined result in another reason for the entailment—but the justification comprising both axioms would violate the minimality condition for justifications.

Cross masking is similar to external masking with the difference that the axioms involved are from other *justifications* for the same entailment rather than non-justification axioms.

Example 2.8.

$$\mathcal{J}_1 = \{A \sqsubseteq C \sqcap D\}$$
$$\mathcal{J}_2 = \{A \sqcup C \sqsubseteq D\}$$

In addition to the two justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ for the entailment $A \sqsubseteq D$, there exists a third reason why the entailment holds, which comprises parts of both justifications: $\{A \sqsubseteq C, C \sqsubseteq D\} \models A \sqsubseteq D$. Again, this third reason, comprising both axioms, cannot be a justification, as it is non-minimal.

Shared cores masking occurs when two *different* justifications have the same reason why an entailment holds, i.e. they only differ in their superfluous parts. This implies that the actual number of explanations for an entailment is *lower* than it appears.

Example 2.9.

$$\mathcal{J}_1 = \{A \sqsubseteq D\}$$
$$\mathcal{J}_2 = \{A \sqsubseteq D \sqcap \exists r.C\}$$

An ontology which contains the two axioms in $\mathcal{J}_1$ and $\mathcal{J}_2$ has two distinct justifications for the entailment $A \sqsubseteq B$. However, the actual reasons why the entailment holds are identical, as the laconic version of J_2 is identical to J_1.

Masking may cause problems for users attempting to understand or repair an entailment. When repairing a justification where internal, external, or cross masking occurs, a user might only notice and modify one of the reasons, expecting the entailment to no longer hold after the modification. That modification, however, may lead to *another* justification for the entailment—an effect which may be rather surprising. Worse even, the justification may interact with several other axioms to create multiple justifications, a situation which has been found to occur in the NCI thesaurus [Hor11a]. Furthermore, when analysing justifications for the purpose of gathering ontology metrics, justification masking may lead to over- or under-counting the number of *actual* justifications in an ontology.

It is clear to see that ontology debugging tools need to support users in cases where masking occurs through non-laconicity. Dealing with superfluity in an

explanation tool, however, makes it necessary to balance two opposing requirements: first, we want the user to be presented with the most concise explanation as to why an entailment holds, and avoid cluttering and distraction caused by superfluous parts. Second, in order to facilitate understanding and repair, the explanations should directly relate to the asserted axioms in the ontology. To date, there have been no successful approaches to OWL ontology debugging that meet both requirements.

Justification-based proofs

In addition to superfluity, there exist other reasons why a justification can be difficult to understand. Justifications may contain structural patterns and constructors whose implications are unfamiliar or non-obvious to the user. A popular example from the *Movie* ontology illustrates how even a very small justification can be hard to impossible to understand for OWL experts [HP09, HPS10b]:

Example 2.10.

$$\begin{aligned}\mathcal{J} = \{&\mathsf{Person} \sqsubseteq \neg\mathsf{Movie},\\ &\mathsf{RRated} \sqsubseteq \mathsf{CatMovie},\\ &\mathsf{CatMovie} \sqsubseteq \mathsf{Movie},\\ &\mathsf{RRated} \equiv \exists\mathsf{hasScript}.\mathsf{ThrillerScript} \sqcup \forall\mathsf{hasViolenceLevel}.\mathsf{High},\\ &\mathsf{domain}(\mathsf{hasViolenceLevel}, \mathsf{Movie})\} \models \mathsf{Person} \sqsubseteq \bot\end{aligned}$$

Example 2.10 shows a justification for the unsatisfiability of the class Person in the *Movie* ontology. Axioms 2 to 5 in the justification entail that $\mathsf{Movie} \equiv \top$, which implies that $\mathsf{Person} \sqsubseteq \mathsf{Movie}$, which, in turn, contradicts the first axiom that states the disjointness of the classes Person and Movie. When presented with this justification, subjects tend to give up or question the correctness of the justification [HPS10b].

Justification-based proofs seek to address the problem of understanding such justifications by providing users with a more detailed explanation which includes not only the justification axioms, but also intermediate entailments such as $\mathsf{Movie} \equiv \top$ in the previous example. Such intermediate entailments, arising from subsets of a justification $\mathcal{J}$ are known as *lemmas*.

In a similar approach to justification-based proofs, Nguyen et al. [NPPW12b] generate natural language proofs based on justifications by identifying frequent sub-patterns, or *rules*, in justifications which lead to such intermediate entailments, then translating the proof trees into natural language. They extracted patterns, restricted to laconic axioms and a maximum size of 4 axioms, from 500 OWL ontologies and identified the 57 most frequent patterns, which are then used as the basis for natural language proofs.

Cognitive complexity

To date, there have only been few attempts at investigating the understandability of justifications for OWL users. While there exists a number of user studies to evaluate OWL debugging tools [KPSH05, Lam07], they focus on measuring the time and success rates of the debugging tools and do not offer any further insights into how users interact with the explanations. The first major study on the cognitive complexity of justifications was carried out by Horridge et al. [HBPS11a] for the purpose of evaluating a complexity model for justifications which was constructed based on an initial exploratory study [HBPS11b]. The study involved 14 students from an MSc class who were presented with a set of items consisting of a set of axioms and an entailment, and were asked to answer whether the entailment followed logically from the axiom set or not. It was found that the complexity predicted by the model coincided partially with the error rate in this task, with some anomalies in the error rates caused by a) superfluity in a justification which the model did not account for, and 2) by a 'flaw' in the experiment protocol which meant test subjects could answer the question correctly, but (presumably) for the wrong reason.

Building on their work on justification-based proofs, Nguyen et al. [NPPW12a] present an investigation of the difficulty users have with understanding various sub-patterns that occur in OWL justifications. The authors conducted a study on a 'mechanical turk' like web platform where test subjects were presented with natural language reasoning problem which was directly based on one of the 51 (out of 57 identified) justification sub-patterns. Each problem was answered by around 50 people, with correct answers ranging from 100% for the easiest rule to only 2% for the most difficult one. While natural language based explanation is potentially highly effective for OWL novices, the resulting proofs are often very large and might be difficult to navigate and understand themselves.

2.3.4 Understanding multiple justifications

In many ontologies used in practice, we find that there exist multiple justifications for a single entailment. While the possible number of justifications per entailment is exponential in the number of axioms in the ontology [BPS07a], the average number of justifications found in OWL ontologies is comparatively small [BHPS11]. However, even for justification counts way below the exponential threshold, we are often faced with several dozen to several hundred justifications for a single entailment. When attempting to repair an ontology, such numbers are clearly not suitable for manual repair by a user; and even with a justification-based repair tool which displays all justifications as a list, there is hardly any chance of producing a minimal repair.

Ji et al. [JQH08] propose a strategy for reducing the number of justifications a user is confronted with by only computing justifications that are the most *relevant* for an entailment. The approach, which was implemented as part of a plugin to the NeOn Toolkit[14] ontology editor, introduces a selection function based on the amount of signature a justification shares with the entailment. While this relevance-based approach may be helpful when users require only a few justifications for *understanding* an entailment, it does not solve the problem of helping a user repair *all* justifications to break an unwanted entailment.

Root and derived justifications

Root and derived justifications provide an additional form of explanation support which aims to assist users in the simultaneous repair of multiple entailments. The idea of unsatisfiable classes depending on the unsatisfiability of another class was originally mentioned in [Sch05a] and explicitly introduced as root and derived unsatisfiable classes [KPSH05]. While the initial focus was on entailed unsatisfiable classes, the concept can be easily extended to arbitrary entailments, as shown in [MMV10]: given a set of entailments $\varepsilon_{\mathcal{O}}$ and their justifications $Justs(\varepsilon_{\mathcal{O}})$, a *derived* justification $\mathcal{J}_D \models \eta_D$ is a justification which is a superset of some other justification $\mathcal{J}, \eta$. Repairing $\mathcal{J}$ first (e.g. by removing or modifying an axiom) will also repair $\mathcal{J}_D$. Note that there may be additional reasons for η_D to hold, which are not covered by this process. A *root* justification is a justification $\mathcal{J}_R$ in $Justs(\varepsilon_{\mathcal{O}})$ which is not a superset of any other justification in $Justs(\varepsilon_{\mathcal{O}})$.

[14] `http://www.neon-toolkit.org/`

Definition 2.6 (Root and derived justifications)**.** *Let $\varepsilon_{\mathcal{O}}$ be a set of entailments of interest, and $\mathcal{J}$ a justification for an entailment η in $\varepsilon_{\mathcal{O}}$. $\mathcal{J}$ is called a* root *justification if there exists no justification $\mathcal{J}'$ for an entailment in $\varepsilon_{\mathcal{O}}$ such that $\mathcal{J}' \subset \mathcal{J}$, else, $\mathcal{J}$ is called a* derived *justification.*

Kalyanpur et al. [KPSH05] carried out the first user study for explanation support in OWL, evaluating the debugging facilities of the ontology editor Swoop. The 12 study subjects were randomly divided into four groups and were given the task to repair unsatisfiable classes in three OWL ontologies using one of three types of debugging tools (clash information from the reasoner and sets of support, root and derived unsatisfiable classes, both), with the fourth group receiving no debugging support at all. Perhaps unsurprisingly, the study found that the group using both the clash information, sets of support (i.e. justifications) and root and derived information performed best (i.e. fastest) in this task.

2.4 Alternative approaches to debugging

Besides justifications (and approaches closely related to the concept of justifications), there exists a number of alternative approaches to ontology debugging. While these may not be as prevalent as justifications in OWL ontology tools, they represent some relevant solutions to the problem of understanding and debugging formal ontologies, such as proofs, natural language techniques, and semi-automated debugging strategies. Under this label, we also gather other approaches to visualising and understanding ontologies, which, while not explicitly tailored towards debugging, may be helpful tools for understanding and repairing entailments.

2.4.1 Proofs

Formal proofs are considered to be the most prevalent alternative form of explanation for logic-based knowledge bases. One of the first approaches to explaining entailments in the CLASSIC system using proof-like structures was presented by McGuinness and Borgida [MB95]. The system omits 'obvious' intermediate steps and provides further filtering strategies in order to generate short and simple explanations. Borgida et al. [BFH99] first introduced a proof-based explanation system for knowledge bases in the Description Logic $\mathcal{ALC}$. The system generates

sequent calculus [Fit83] style proofs using an extension of a tableaux reasoning algorithm, which are then enriched to create natural language explanations. Aiming to 'provide the simplest explanation possible', the authors also introduce a *relevance* function which simplifies the proofs by omitting 'irrelevant' parts. The consensus of these proof-based approaches seems to be that good proofs ought to be 'as simple and short as possible', yet, there have been no formal investigations into how ontology developers interact with such proof-based explanations.

While proof presentation in other logics has been well studied (e.g. [FH88]), surprisingly few authors are concerned with the cognitive aspects of proof understanding. There exist formal definitions of *obvious proof steps* [Dav81, Rud87], however, these are merely formulated with the goal of creating *short* proofs and do not investigate other sources of complexity for human readers. By comparison, Lingenfelder [Lin89] also considers the skill level of the proof reader to be crucial in order to determine what can be considered 'trivial' in a proof, and emphasises the need for a *user model* in order to provide suitable proofs.

2.4.2 Ontology revision

Ontology revision as described by Nikitina et al. [NRG12] follows a semi-automated approach to ontology repair, with a focus on factually incorrect statements rather than logical errors. In the ontology revision process, a domain experts inspects the set of ontology axioms, then decides whether the axiom is correct (should be accepted) or incorrect (axiom is rejected). Each decision thereby has consequences for other axioms, as they can be either automatically accepted (if they follow logically from the accepted axioms) or rejected (if they violate the already accepted axioms).

The ontology revision system determines the impact a decision has on the remainder of the axioms (using a ranking function), and presents high impact items first in order to minimize the number of decisions a user has to make. Conceptually, this approach is straightforward and easily understandable for a user, as the cognitive effort is reduced to a simple yes/no decision, and the tool attempts to minimize the number of decisions that need to be made. In order to debug unwanted entailments, e.g. unsatisfiable classes, the set of unwanted consequences can be initialised with those erroneous axioms. The accept/decline decisions are then made in order to remove those axioms which lead to the unwanted entailments.

2.4.3 Direct computation of diagnoses

The direct diagnoses computation approach [FS05, DS08, SFRF12, RSFF12] is directly related to justifications, but rather than computing the set of justifications for an entailments which is then repaired by repairing or modifying a minimal hitting set of those justifications, the diagnoses (i.e. minimal hitting sets) are computed directly. The direct computation of diagnoses using the hitting set tree algorithm was first proposed by Friedrich and Shchekotykhin [FS05]. The authors argue that debugging large numbers of conflicts and potentially diagnoses by computing justifications poses a computational challenge, while direct computation of diagnoses provides a more efficient solution.

Both the ontology revision approach by Nikitina et al. and the direct computation of diagnoses show some advantages over computing justifications in terms of computational performance. However, the semi-automated repair presented in [NRG12] means that the user has no direct control over which axioms to remove or modify in order to repair the unwanted entailments, while the direct computation of diagnoses aims exclusively at finding a minimal repair, regardless of the factual correctness or incorrectness of the axioms involved in the diagnoses. Furthermore, neither of the approaches supports *understanding* why those entailments hold, as the users are not presented with the actual *reasons*, but only see a sequence of axioms or a set of diagnoses, respectively. While both approaches aim to reduce the total number of steps taken in the debugging process, they might still require a user to go through a long and tedious revision process when repairing a large number of errors.

2.4.4 OntoClean

OntoClean [GW02, GW04] provides a framework for making modelling decisions in the ontology development process, with a strong focus on correct subumption relationships. This is motivated by a misuse of subsumptions to express other relations than is-a relationships, for example part/whole relationships. OntoClean helps users to validate taxonomies by identifying *metaproperties* of each class, such as *rigidity*, *identity*, *unity*, and *dependency*. The framework then specifies constraints on subsumptions between classes with different properties, such as if a class A is anti-rigid, i.e. an instance of A can *cease* to be a member of the class,

then every subclass of A must also be anti-rigid. By applying OntoClean guidelines when building taxonomies, ontology developers can prevent subsumption relationships which may later lead to modelling inconsistencies.

2.4.5 Ontology comprehension

Ontology users often require support in understanding an ontology, or parts of it, for example, when integrating an existing ontology into a project, which requires the ontology developer to familiarise themselves with the structure of the adopted ontology. While not specifically aimed at ontology debugging, such comprehension and visualisation tools can help users to get a better grasp of the relationships between entities in an ontology, which may provide support in the debugging process.

Ontology visualisation tools, such as the OWLViz plugin[15] in Protégé 4, the CropCircles tool [WP06] and the, admittedly rather exotic, music score notation of Barzdins and Barinskis [BB07], offer ontology users support when exploring an ontology in order to gain an overview of the relationships between its classes.

Visualisation techniques can also be used for the purpose of *model exploration* [BSP09], which can user support in understanding *non-entailment*. Users may want to understand, for instance, why a class is not entailed to be subclass of some other—a fact that cannot be explained by justifications, as it is not possible to point out a particular subset of the ontology that does *not* entail something. The idea underlying model exploration is that seeing (subsets of) the models of an ontology helps users understand the relations between its classes, which, in turn, may support them in understanding why an entailment holds or does *not* hold in the ontology.

2.5 Summary and conclusions

In this chapter, we have laid out the foundations for the work presented in this thesis. We introduced the basic concepts of description logic knowledge bases, such as their syntax, semantics, and standard reasoning services, and introduced the Web Ontology Language OWL. We discussed the scope of logical and non-logical errors which can occur in OWL ontologies and introduced justifications

[15] `http://www.co-ode.org/downloads/owlviz/`

as the currently dominant form of debugging support for entailments of OWL ontologies. This was followed by a closer look at the strategies for computing justifications, as well as some of the applications of justifications for repairing errors in OWL ontologies. Finally, we gave an overview over debugging strategies for description logic knowledge bases which are not directly related to justifications or only make indirect use of them, finding that the majority of approaches do not consider cognitive aspects of how users interact with explanation.

While multiple justifications for entailments of OWL ontologies have been acknowledged as a phenomenon to be found in OWL ontologies, the main focus of research thus far has been the efficiency of computing multiple justifications. In contrast, the issue of how users can find a suitable repair once those justifications have been computed has been largely neglected. There exist some approaches to making individual justifications easier to understand, such as laconic justifications and justification-based proofs, but the specific problem of debugging and repair in the presence of multiple justifications has not been addressed in prior research, except for some work looking at root and derived unsatisfiable classes. Root and derived unsatisfiable classes provide a first level of additional support for a particular relationship between multiple justifications for multiple entailments; yet, there have been no investigations into whether root/derived relationships are indeed a common feature of OWL justifications, what steps can be taken when this feature is not present, and how users can cope with multiple justifications for *single* entailments. Besides, while root and derived unsatisfiable classes indicate that there exist structural relationships between justifications, there have not been any investigations into other such relationships and whether they can be exploited for debugging.

In summary, there does not seem to be a good understanding of how prevalent multiple justifications are in ontologies used in practice, neither are there sophisticated enough coping strategies to help ontology engineers deal with multiple justifications efficiently. These insights motivate the research presented in this thesis, as there is an obvious need for further investigation of the occurrence of multiple justifications, the relations between them, and techniques for reducing both mechanical (the number of steps taken to achieve a goal) as well as mental (the cognitive complexity of the information presented to a user) user effort in the presence of multiple justifications.

Chapter 3

Defining finite entailment sets

The justificatory structure of an OWL ontology $\mathcal{O}$ describes the relationships between the justifications for *some* set of entailments of $\mathcal{O}$. However, if we want to compute the set of justifications for 'the entailments' of an ontology, we immediately encounter a problem: recall that the set of entailments of an ontology $\mathcal{O}$ is the set of all axioms α such that $\mathcal{O} \models \alpha$—a set which is infinite for OWL ontologies, and in most cases it does not seem practical to compute and analyse justifications (other than the empty ontology over a non-empty signature) for an infinite set of entailments.

Thus, we need to restrict the set of entailments we analyse to some *finite* subset of the deductive closure of $\mathcal{O}$. Classification of an ontology, that is, computing all entailed subsumptions and equivalences containing only named classes, is a standard reasoning task, and the resulting class hierarchy (in the form of either the atomic subsumption axioms or both subsumption and equivalence axioms) of the ontology is a commonly used entailment set in OWL applications.

However, even if we restrict our attention to the class hierarchy of an ontology, we find that there is no standard way of *representing* this set of axioms. Worse even, misleading nomenclature in ontology tools, such as the Protégé 4 editor and the OWL API, as well as anecdotal evidence show that there exist common misconceptions about the entailments: it is often assumed that only non-trivial information is contained in the set of entailments, and tautologies such as $A \sqsubseteq A$ are not 'real' entailments. On the other hand, however, some OWL tools *do* include tautologies of the type $A \sqsubseteq \top$ for certain named classes. And finally, the term 'entailments' is frequently used to refer to information which is only *inferred* but not asserted in the ontology, leading to asserted axioms not being considered entailments.

This means that, despite having a well-defined finite set of entailments at hand (the set of all axioms α of type $A \sqsubseteq B$ and, if required, $A \equiv B$ for $A, B \in sig(\mathcal{O})$ such that $\mathcal{O} \models \alpha$), we still obtain different numbers of entailments for

the same ontology depending on the representation chosen by the application we are dealing with. And while it may not be necessary to define a single canonical representation of the class hierarchy—after all, axioms such as $A \sqsubseteq \top$ *can* be useful for some applications while they might be superfluous in others—we need to at least make obvious the different design decisions to be made when choosing a representation for this entailment set.

In what follows, we will discuss various design decisions for representing finite entailment sets of consistent OWL ontologies, which takes into consideration relationships such as direct and indirect subsumptions, tautologies, dealing with unsatisfiable and universal classes, and ontology imports, while paying attention to the issue of *counting* entailments. The design decisions presented here largely focus on the class hierarchy of an ontology, but are also applicable to arbitrary entailments, and we will also discuss how we can define finite entailment sets *beyond* the simple class hierarchy of an ontology. Finally, we introduce a shorthand notation which allows us to conveniently refer to a specific representation of an entailment set, and outline some examples of how entailment sets are used in OWL applications.

While this chapter informs the experiments on justifications presented in the remainder of this thesis, it may also be regarded as a self-standing discussion with relevance to applications such as ontology analysis, the OWL API, and ontology editing tools.

3.1 Design decisions for finite entailment sets

In this section, we will describe a number of design decisions to make when choosing a representation for a finite entailment set of an OWL ontology. In other words, given a finite set of entailments, such as the axioms representing the class hierarchy, we suggest a set of filtering steps which result in an axiom set that is logically equivalent, but may differ in the number of axioms it contains.

To clarify the distinction between a finite entailment set and its representation, consider a simple example: given an ontology $\mathcal{O} = \{A \sqsubseteq B, B \sqsubseteq C\}$, we are interested in all entailed atomic subsumptions of type $X \sqsubseteq Y$ where $X \neq Y$, $X \neq \bot$, and $Y \neq \top$. This gives us the finite entailment set $\varepsilon_{\mathcal{O}} = \{A \sqsubseteq B, B \sqsubseteq C, A \sqsubseteq C\}$. However, we may also choose to *represent* $\varepsilon_{\mathcal{O}}$ by an equivalent, more compact set of axioms $\varepsilon_{\mathcal{O}}' = \{A \sqsubseteq B, B \sqsubseteq C\}$, which is logically equivalent to

$\varepsilon_{\mathcal{O}}$, but contains fewer axioms.

Only the choice of entailment types has an effect on the actual *size* of a finite entailment set: given two ontologies $\mathcal{O}$ and $\mathcal{O}'$ such that $\mathcal{O} \subset \mathcal{O}'$, we expect the number of axioms in a finite entailment set of the *larger* ontology $\mathcal{O}'$ to be the same or more than that of $\mathcal{O}$. That is, our notion of finite entailment sets must be *stable* under the extension of an ontology $\mathcal{O}$; an essential criterion when using entailment counts for the comparison of two ontologies.

However, as we will see in the following sections, some design decisions for *representations* of entailment sets appear to violate the stability of the entailment set relation by leading to *smaller* (representations of) entailment sets when *adding* axioms to an ontology. Where applicable, we will point out the impact of a decision on the stability of entailment counts.

3.1.1 Tautologies

A tautology is an axiom such as $A \sqsubseteq A$, $A \sqsubseteq \top$, and $\bot \sqsubseteq A$ for a named class $A \in sig(\mathcal{O})$ which is vacuously entailed by $\mathcal{O}$. Tautologies do not contain any relevant information and are therefore frequently omitted from the axiom set representing the class hierarchy of an ontology.

However, in some OWL tools, such as the OWL API and Protégé 4, named classes that do not have a direct named subsumer are generally *included* in entailment sets as subclasses of $\top$, in order to ensure that all named classes in the ontology are present in the entailment set in some way. Note that this strategy means that *only* classes which do not have a named subsumer occur in axioms of the type $A \sqsubseteq \top$. This can be problematic if we use the entailment count to compare the logical strength of two ontologies: take, for instance, two ontologies over the signature $\{A, B, C\}$ where $\mathcal{O} = \emptyset$ and $\mathcal{O}' = \{A \sqsubseteq B\}$. If we include axioms of the above type in the entailment set, we get the two entailment sets

$$\begin{aligned}\varepsilon_{\mathcal{O}} =& \{A \sqsubseteq \top, B \sqsubseteq \top, C \sqsubseteq \top\}\\ \varepsilon_{\mathcal{O}'} =& \{A \sqsubseteq B, B \sqsubseteq \top, C \sqsubseteq \top\}.\end{aligned}$$

Both entailment sets have the same number of entailments; however, $\varepsilon_{\mathcal{O}'}$ arguably contains at least *some* non-trivial information (namely the subsumption $A \sqsubseteq B$). While this decision does not affect the monotonicity of the entailment count, it shows that, if we include tautologies in an entailment set, the size of the set may

not reflect the amount of relevant information it contains.

Since there is no obvious benefit to including most tautologies in a finite entailment set, we will generally consider finite entailment sets to be free of tautologies in the remainder of this chapter. If required, we may only include axioms of the type $A \sqsubseteq \top$ for a named class A which does not have any other named subsumer.

3.1.2 Asserted and inferred axioms

Every axiom that is asserted in an ontology $\mathcal{O}$ is trivially entailed by the ontology and therefore part of the set of entailments of $\mathcal{O}$. However, the main idea behind logic-based ontologies is the application of automated reasoning to make *implicit* knowledge visible; thus, the set of *inferred*, but not asserted, axioms may be of much higher relevance to a user. In particular when counting entailments of an ontology, including asserted axioms in the count may not add any useful information if the number of asserted axioms of the selected type is already known. Most OWL tools (e.g. Protégé 4 and the OWL API) take this into account when representing the class hierarchy of an ontology and return only entailments which are inferred, but not asserted.

While excluding asserted axioms from an entailment set is a reasonable choice in user-focused applications, this may lead to a loss of information in analytical applications, such as justification analysis. An entailment that is asserted in an ontology may also hold for other, more complex reasons, i.e. it can have justifications other than the asserted axiom itself. This may be purely accidental and a side-effect of other axioms in the ontology, or intentional as a result of ontology developers 'adding' inferred entailments into the ontology. Making entailments explicit has two advantages: first, it may improve reasoner performance when classifying the ontology, as the reasoner will have to perform fewer subsumption checks, and second, the information will be visible even when there is no reasoner available, for example in a web-based ontology browser—and even if these are not real benefits, users may *believe* that they are, which encourages them to take these steps.

When counting entailments, we need to be aware of the effects of excluding asserted entailments from the entailment set. Assume an ontology $\mathcal{O}$ with n asserted atomic subsumptions and $k \geq 1$ inferred (but not asserted) atomic subsumptions. If we only count entailments which are inferred but not asserted, the

entailment count for $\mathcal{O}$ will be k. Now, if we add all k inferred atomic subsumption axioms to $\mathcal{O}$, we obtain $\mathcal{O}'$ such that $\mathcal{O} \subset \mathcal{O}'$. Then the entailment set of $\mathcal{O}'$ will contain no (inferred but not asserted) axioms, that is, $\varepsilon_{\mathcal{O}'} \subset \varepsilon_{\mathcal{O}}$ despite $\mathcal{O} \subset \mathcal{O}'$. This is obviously counter-intuitive, as we generally expect the number of entailments to grow monotonically as we add axioms to an ontology.

A more concrete example shows how the distinction between inferred and asserted entailment makes the entailment count dependent on the syntactical variations of equivalent axioms: take $\mathcal{O} = \{A \sqsubseteq B \sqcap C\}$ and $\mathcal{O}' = \{A \sqsubseteq B, A \sqsubseteq C\}$. $\mathcal{O}$ has two entailed atomic subsumptions ($A \sqsubseteq B$, $A \sqsubseteq C$) which are inferred and not asserted, whereas $\mathcal{O}'$, despite having the same entailed atomic subsumptions, has no entailments according to our specification.

Of course, in both examples the 'missing' entailments are only a deception, as the two ontologies are logically equivalent, which means they do not actually differ in terms of their information content; however, if we use that the number of inferred (and not asserted) entailments as an ontology metric to compare the two ontologies, we can easily see how this situation can quickly lead to counting errors.

3.1.3 Transitivity

We generally treat the class hierarchy of an ontology as a directed graph, with nodes representing sets of equivalent classes, and edges representing subsumption relationships. A *direct* subsumption between a class A and a class B is represented by an edge between the nodes labelled with A and B, respectively; this corresponds to a path of size one. An *indirect* subsumption is a path between two nodes of size greater than one. The class graph can be either based on the *asserted* ontology, i.e. the class graph is constructed based on the axioms in the ontology, without the use of a reasoner, or the *inferred* ontology, which means that the class graph is built according to the subsumptions and equivalences returned by a reasoner. That is, a directed edge between the nodes labelled A and B is added to the class graph of $\mathcal{O}$ if $A \sqsubseteq B$ is asserted in (inferred by) $\mathcal{O}$, and a class name C is added to the label of a node labelled $\{D\}$ if $C \equiv D$ is asserted in (inferred by) $\mathcal{O}$. Note that in Protégé 4, the class hierarchy display is generated using a very simple *structural* reasoner which, in addition to implementing transitivity of the subsumption relationship, simply splits conjunctions on the RHS of subsumption axioms and in equivalent classes axioms.

The set of all entailed atomic subsumptions, direct and indirect, then corresponds to the transitive *closure* of the inferred class graph, while the transitive *reduct* [AGU72] represents the set of entailed *direct* subsumptions. The transitive reduct of a directed acyclic graph G is a graph G' such that G' has a directed path between two nodes u, v whenever there is a path (u, v) in G, and there is no graph with fewer edges than G' which fulfils that condition. The abstract concept of a transitive reduct can be represented as the Hasse diagram[1] of an ontology's class graph.

There are several scenarios in which we may want to include or exclude indirect subsumptions in a representation of the class hierarchy. When presenting a set of entailed axioms to a user, we may assume that the user understands the principle of transitivity and only requires a small entailment set with *relevant* information; in such a case, it seems reasonable to choose a compact representation of the entailment set by excluding indirect subsumptions. On the other hand, if a user wants to be presented with *all* the information entailed by an ontology, it might be preferable to include indirect subsumptions. For the purpose of computing and analysing justifications of an ontology, for example, we may want to include indirect subsumptions in the entailment set, as they may have relevant justifications which would otherwise be neglected.

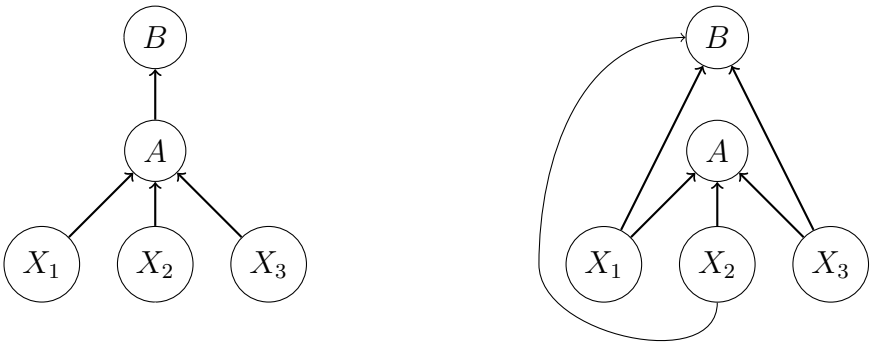
With regards to counting the number of entailments of an ontology, care needs to be taken when dealing with the transitive reduct of the inferred class graph. Example 3.1 shows how *removing* axioms from an ontology results in an *increase* in the number of entailments.

Example 3.1.

$$\mathcal{O} = \{X_1 \sqsubseteq A, X_2 \sqsubseteq A, X_3 \sqsubseteq A \\ \mathsf{X_1} \sqsubseteq \mathsf{B}, X_2 \sqsubseteq B, X_3 \sqsubseteq B, A \sqsubseteq B\}$$

Given the above ontology, the number of entailed direct atomic subsumptions, based on the transitive reduct of the inferred class graph (shown in Figure 3.1a), is 4, as the indirect subsumptions between X_i and B are excluded from the set: $\varepsilon_{\mathcal{O}} = \{X_1 \sqsubseteq A, X_2 \sqsubseteq A, X_3 \sqsubseteq A, A \sqsubseteq B\}$. Removing the axiom $A \sqsubseteq B$ from

[1] See, for example, [JGSV06] for an example of diagrammatic representations of an OWL ontology.



(a) Transitive reduct of $\mathcal{O}$.

(b) Transitive reduct of $\mathcal{O}'$.

the ontology creates the class graph for the now modified ontology $\mathcal{O}'$ shown in Figure 3.1b; as we can see, the transitive reduct of the graph has *more* edges than the graph for $\mathcal{O}$. Accordingly, the set of entailments based on the transitive reduct now contains 6 entailed axioms: $\varepsilon_{\mathcal{O}'} = \{X_1 \sqsubseteq A, X_2 \sqsubseteq A, X_3 \sqsubseteq A, X_1 \sqsubseteq B, X_2 \sqsubseteq B, X_3 \sqsubseteq B\}$. Thus, the removal of an axiom has led to a seemingly *larger* entailment set in this particular representation.

Interestingly, the `InferredSubClassAxiomGenerator` implemented by the OWL API exhibits exactly this behaviour; and while this non-monotonicity of entailment counts may be technically correct, it is certainly confusing and counter-intuitive to a user who has no way of specifying and understanding the exact settings of an entailment generator.

3.1.4 Equivalent classes

While we commonly only deal with entailed atomic subsumptions representing the class hierarchy, we might also want to include information about the equivalence between named classes in the ontology in an entailment set. However, dealing with equivalent classes (or properties) axioms as entailments is not as straightforward as it may seem, as choosing how to represent equivalent classes as well as subsumptions between equivalent classes and other classes as a set of axioms requires us to make a number of design decisions.

Representing equivalence as axioms Assume some ontology $\mathcal{O}$ entails the equivalence of the classes A, B, and C. In the class graph of $\mathcal{O}$ this can be easily expressed by a node which is labelled with the three class names. In OWL applications, however, users are generally presented with a list of entailed *axioms*

rather than a graph visualisation. We now have to choose how to represent the equivalence between these three classes in a set of equivalent classes axioms:

1. As an n-ary *EquivalentClasses* axiom, which is possible in OWL.
2. As an exhaustive set of binary equivalent classes axioms in order to correspond to DL notation which only allows binary equivalent classes axioms
3. As a minimal set of binary equivalent classes axioms. This requires the use of a function $Pairwise(n)$ to return a set of *pairwise* axioms which suffices to represent the relation, such as $\{A \equiv B, B \equiv C\}$. This method is implemented in the OWL API.

Again, the decision how to represent equivalent classes depends on the application of the entailment set. In an OWL context, an n-ary EquivalentClasses axiom is certainly the most user-friendly way of representing a set of equivalent classes. When analysing justifications, an exhaustive set of binary equivalent class axioms will capture all justifications, while a set of representative axioms may be suitable in a user-facing application which only supports binary equivalent classes axioms.

Subsumptions between equivalent classes Representing the subsumption relationships in a class graph between nodes labelled with multiple equivalent classes poses yet another challenge: given a set of equivalent classes A, B, and C which are all subsumed by a common superclass D, how can we represent this relation in a set of axioms?

Yet again, there are multiple approaches, which depend on whether an entailment set explicitly contains equivalence classes axioms, or whether the equivalence between (sets of) classes may be represented otherwise:

1. Every class in the set of equivalent classes creates a new subsumption axiom $A \sqsubseteq D$, $B \sqsubseteq D$, $C \sqsubseteq D$. If the chosen entailment set does *not* include equivalent class axioms, this approach may be suitable, as it explicitly lists the subsumption relations between all classes and their superclasses.[2] The disadvantage of this strategy is a fast growth of subsumption axioms if both the subclass and the superclass node contain multiple equivalent classes. That is, for two nodes labelled with i and j equivalent classes, respectively, the number of axioms resulting from this approach is $i * j$.

[2]Excluding the equivalent class axioms in this situation may lead a user to have an incorrect 'mental model' of the ontology graph in which the three subclasses are not equivalent.

2. When including equivalent class axioms in the entailment set, it may suffice to select a *representative* from each of the nodes representing the sub- and superclasses, respectively. For this purpose, we introduce a new function, $Rep(n)$ which selects a class name from a node n of equivalent classes based on some user-defined criteria. The knowledge of the equivalence relations and the subsumption between two of the classes will then be sufficient information for a user to infer that the other subsumptions follow.
3. Another, less straightforward approach, would be to attempt to represent all equivalent classes in a node by one newly generated class name, for example by concatenating the names of the subclasses and presenting the user an axiom of the type $A, B, C \sqsubseteq D$. This is obviously not standard OWL notation and might be confusing to a user, yet, it conveniently captures both the equivalence as well as the subsumption relations in a single axiom.

3.1.5 Strict and non-strict subsumptions

Another issue to consider when computing finite entailment sets is the notion of representing strict vs non-strict subsumptions. A strict subsumption is an axiom $A \sqsubseteq B$ such that $\mathcal{O} \models A \sqsubseteq B$ and $\mathcal{O} \not\models B \sqsubseteq A$, i.e. the classes are not equivalent, whereas a non-strict subsumption is an axiom $A' \sqsubseteq B'$ or $B' \sqsubseteq A'$ where $\mathcal{O} \models A' \equiv B'$.

By default, OWL reasoners (when accessed via the OWL API) exclude non-strict subsumptions when asking for all superclasses of a named class. This may cause confusion in cases where a user knows that class A should be subsumed by B, but is not aware of reasons for the equivalence of the two classes; the subsumption simply appears to be missing from the entailment set. Including non-strict subsumptions therefore seems reasonable when a user requests only subsumption axioms and no equivalent classes axioms in an entailment set. On the other hand, when including equivalence axioms in an entailment set, non-strict subsumptions should in turn be excluded from the set in order to avoid duplicating information in the resulting set.

When counting entailments, we are faced with yet another situation in which the number of entailments does not grow monotonically in the size of the ontology. Take the two ontologies $\mathcal{O} = \{A \sqsubseteq B\}$ and $\mathcal{O}' = \{A \sqsubseteq B, B \sqsubseteq A\}$. $\mathcal{O}'$ clearly contains more axioms and is stronger than $\mathcal{O}$, however, if we count only strict subsumptions in the entailment set (as is the default in the OWL API), the

number of entailments of $\mathcal{O}'$ is less than that of $\mathcal{O}$. As in the previous examples, the *actual* number of entailments of $\mathcal{O}'$ as given by the entailment relation $\models$ is greater than that of $\mathcal{O}$, however, our entailment sets *appear* to violate the monotonicity of $\models$.

3.1.6 Equivalence to top and bottom

An unsatisfiable class in an ontology is a class which is equivalent to $\bot$, i.e. it is mapped to the empty set in all interpretations. This also means that any unsatisfiable class is a subclass of any other class in the ontology. In ontology editors and in conversational use, however, unsatisfiable classes are frequently denoted as *subclasses of bottom* and no subsumptions between the unsatisfiable class and named classes are included in entailment sets. For example, the OWL API's `InferredSubClassOfAxiomGenerator` only returns atomic subsumptions involving *satisfiable* classes.

Similarly, a class which is entailed to be equivalent to $\top$ (we may call it a *tautological* class) is commonly denoted as *superclass of* $\top$, and is entailed to be a superclass of any other named class in the ontology. Interestingly, there is no symmetry between the treatment of unsatisfiable and tautological classes, as entailments of the type $B \sqsubseteq A$ for a tautological class $A \equiv \top$ and a named class B *are* commonly displayed in ontology editors and returned by the OWL API.

Regarding the naming conventions, while these are obviously not incorrect, they only show one direction of the equivalence relationship between the classes which may be misleading to users; furthermore, if we exclude non-strict subsumptions from an entailment set in a principled manner, we should also exclude axioms of the type $A \sqsubseteq \bot$ and $\top \sqsubseteq A$, as these are non-strict subsumptions. On the other hand, for both $\top$ and $\bot$ the subsumption in one direction ($\bot \sqsubseteq A$, $A \sqsubseteq \top$) is in fact trivial, and we are really only interested in the other direction of the equivalence. This is why it seems reasonable to treat equivalences with $\top$ and $\bot$ separately from named classes and generally display them as subsumption axioms.

With respect to the different treatment of unsatisfiable and tautological classes, it is clear to see that a tautological class can introduce more subtle errors in the ontology, while subsumptions involving unsatisfiable classes do not contain any relevant information. Take, for instance, the *Movie* ontology example which we have mentioned in Chapter 2. In this ontology, the class Person is entailed to be

unsatisfiable. This is caused by the fact that the class Movie is tautological (i.e. 'everything is a Movie '), but asserted to be disjoint with Person. Understanding that Movie is a superclass of Person is crucial to understanding the cause of the error. This example shows how (direct or indirect) subsumptions involving tautological classes add important information to an entailment set and should therefore always be included in an entailment set.

3.1.7 Axiom and expression types

Thus far we have focused on the design decisions which affect the representation of the class hierarchy of an ontology. However, it is clear to see that an application might require a finite entailment set which contains axiom and expression types beyond atomic subsumptions and equivalences. For example, it can be useful to know how many named classes are subsumed by an expression such as $\exists r.B$ for an arbitrary named property r and class B, as this can justify the introduction of a named class for the anonymous expression. This means that we need some way of restricting the infinite set of entailments of an OWL ontology to a sensible finite set, which strikes a balance between containing large amounts of irrelevant information and omitting relevant axioms.

Prime implicates [Qui52, Qui59, Jac92, Bie09] fulfil the requirement of providing a finite representation of the set of entailments of a formula. Initially defined for propositional formulae, Bienvenu [Bie07] extends the notion of prime implicates to concept expressions in the description logic $\mathcal{ALC}$. First, literals L, clauses Cl, and cubal concepts Cb are defined as follows:

$$\begin{aligned} L &::= \top \mid \bot \mid A \mid \neg A \mid \forall r.Cl \mid \exists r.Cb \\ Cl &::= L \mid Cl \sqcup Cl \\ Cb &::= L \mid Cb \sqcap Cb \end{aligned}$$

A clausal concept Cl is then a prime implicate of a concept expression C if

1. $\models C \sqsubseteq Cl$
2. For any Cl' such that $\models Cl' \sqsubseteq Cl$, then $\models Cl \sqsubseteq Cl'$.

Prime implicates are the 'logically strongest clausal consequences' [Bie09] of a formula that do not contain any redundancies; removing any literal from a prime implicate would cause the clause to no longer be entailed. Prime implicates of this type do not correspond to our notion of entailments as sets of axioms, but

rather describe subsumptions between expressions. We can obtain a finite set of axioms, for example, by generating the set of class assertions $x : Cl$ for all $x : C$, or the set of subsumption axioms $X \sqsubseteq Cl$ for all $X \sqsubseteq C$ for $x, X \in sig(O)$. As an example [Bie07], take the following expression:

$$A \sqcap (B \sqcup C) \sqcap \exists r.\top \sqcap \forall r.(B \sqcap (A \sqcup C)) \sqcap \forall r.(B \sqcup D)$$

The prime implicates of this expression are the expressions $A, B \sqcup C, \exists r.(B \sqcap A) \sqcup \exists r.(B \sqcap C), \forall r.B, \forall r.(A \sqcup C)$.

While prime implicates provide an elegant way of defining a 'natural' finite set of entailments, the concept has only been extended to concept expressions in $\mathcal{ALC}$ and does not cover the full spectrum of constructors and axioms available in OWL. We therefore look at simpler, purely syntactical strategies for generating finite entailment sets.

Similar to restricting the finite entailment set to entailed atomic subsumptions, we can specify any other axiom type as well as expression types we want to consider by providing an entailment pattern. In order to generate entailed axioms of different types containing only named classes, the OWL API provides an `InferredAxiomGenerator` interface whose implementations provide access to generators for *all* OWL axiom types, such as `InferredDisjointAxiomGenerator` and `InferredSubDataPropertyAxiomGenerator`.

Beyond atomic entailments, we may also want to include entailments involving *complex* expressions, such as existential or universal quantifiers. Entailments containing complex expressions are currently not supported 'out-of-the-box' by tools such as the OWL API; however, it is certainly possible to programmatically generate axioms containing complex expressions over the signature of an ontology $\mathcal{O}$, check whether these are entailed by $\mathcal{O}$, and if yes, include them in the entailment set.

In order to guarantee that the entailment set generated that way is indeed finite, we need to impose restrictions on the types and nesting depths of expressions. For example, we may only include axioms which do not contain any nested expressions, such as $A \sqsubseteq \exists r.B$, $A \sqsubseteq \forall r.B$, $A \sqsubseteq B \sqcap C$, $A \sqsubseteq B \sqcup C$, $A \sqsubseteq \neg B$. Finally, once these complex entailments have been specified, the design decisions we have discussed thus far can be applied in order to choose a suitable representation of the set.

3.1.8 Dealing with ontology imports

Another issue that needs to be dealt with when extracting and counting entailments from an ontology is its import structure. An OWL ontology $\mathcal{O}$ that imports another OWL ontology $\mathcal{O}'$ can have different kinds of entailments: those that hold in $\mathcal{O} \setminus \mathcal{O}'$ (*native* entailments), those that are entirely from the imported ontology, i.e. they hold in $\mathcal{O}' \setminus \mathcal{O}$ (*imported* entailments), and those that are neither native, nor imported, and hold in $\mathcal{O} \cup \mathcal{O}'$ (*mixed* entailments).

When performing analytical tasks on a corpus of ontologies, disregarding the issue of imported entailments may in fact lead to significant distortions: imagine a scenario where each ontology $\mathcal{O}_i$ in a test corpus imports another ontology $\mathcal{O}'$, for instance an upper-level ontology. A finite entailment set of each of $\mathcal{O}_i$ would also include all imported entailments of $\mathcal{O}'$, which would skew any analysis of the corpus towards the entailments of $\mathcal{O}'$. We encountered this situation when analysing a snapshot of the NCBO BioPortal[3] corpus, in which a number of ontologies had exactly the same set of entailments, as they were all importing the *Basic Formal Ontology*[4] (BFO).

In order to resolve this issue, we propose a classification of entailment types based on the *origin* of an entailment which is determined by the set of its justifications. We use the following naming conventions:[5]

- $\mathcal{O}_{root}$ denotes the ontology *document* we are analysing, e.g. the .owl file that has been loaded into an ontology editor.
- $\mathcal{O}$ is the *import closure* of $\mathcal{O}_{root}$, i.e. the ontology resulting from transitive closure of the imports of $\mathcal{O}_{root}$.
- $\mathcal{O}'$ denotes an ontology in the import closure of $\mathcal{O}_{root}$ that is not the root ontology itself.

Intuitively, a native entailment originates entirely from the root ontology, that is, all its justifications contain only axioms from the root $\mathcal{O}_{root}$. An imported entailment originates entirely from entailments that are *not* from the root ontology, whereas a *mixed* entailment covers the remaining scenarios:

Definition 3.1. *A justification $\mathcal{J}$ for an entailment η is a*

1. native *justification in $\mathcal{O}$ if $\mathcal{J} \in \mathcal{O}_{root}$.*
2. imported *justification in $\mathcal{O}$ if $\mathcal{J} \cap \mathcal{O}_{root} = \emptyset$.*

[3] `http://bioportal.bioontology.org/`
[4] `http://www.ifomis.org/bfo`
[5] As used in `http://www.w3.org/TR/owl2-syntax/#Imports`

3. mixed *justification otherwise.*

An entailment η is called

1. native *if it has* only *native justifications.*

2. imported *if it has* only *imported justifications.*

3. mixed *otherwise.*

We can see straight away that there are several constellations of mixed entailments having native, imported, and mixed justifications. While such a fine-grained distinction may not be relevant for analytical purposes (it seems more reasonable to simply include *all* entailments of a specified type from an ontology $\mathcal{O}$ and its imports closure), we list them here for completeness:

1. Purely mixed: An entailment which has *only* mixed justifications.
2. Partially mixed: An entailment which has
   a) native and imported justifications.
   b) mixed and native justifications.
   c) mixed and imported justifications.
   d) mixed, native, and imported justifications.

3.2 A notation for finite entailment sets

Based on the design decisions we have discussed in the previous section, we will now introduce a shorthand notation for the various aspects of finite entailment sets. The notation will allow us to conveniently refer to specific entailment sets, for example, in OWL tools, the OWL API, and in experiment descriptions. Further, we will also introduce notations for *wanted* and *unwanted* entailment sets to denote partitions of a given finite entailment set. The section is concluded by several examples of entailment sets to demonstrate how the notation we have introduced can be applied in practice.

3.2.1 Introducing the notation

Table 3.1 lists the keys and polarities (+ or -) we assign to the individual design decisions and the defaults. Note the following remarks:

1. The keys for the properties in the top section of the table are constructed to always assume the *negative* polarity key as the default where applicable, e.g. T^-. The negative polarity is given simply for completeness reasons and

to specify the default; if clear from the context, we choose to simply drop the key to assume the default option.

2. 1. implies that when *including* a certain type of entailment, the + sign indicating positive polarity of the key is not necessarily required; e.g. we could simply write T to indicate the presence of $A \sqsubseteq \top$ type axioms. However, we choose to always include the polarities in order to make the decision explicit and avoid ambiguity.
3. The exception with regards to polarity is the inclusion of native entailments by default, indicated by the positive key I^{n+}.
4. Keys and polarities can be grouped, e.g. to describe the set of atomic subsumptions including asserted axioms and indirect subsumptions, we write 'the $(AD)^+$ set of axioms of the type $A \sqsubseteq B$ for all named, satisfiable classes A, B in $\mathcal{O}$ '.
5. The decision whether to include unsatisfiable and tautological classes as subsumption or equivalent classes axioms is omitted from the table, as it is part of the description of the axiom and expression types in an entailment set specification.
6. The keys E^r and N^r require the specification of a function *Pairwise* and *Rep*, respectively.

3.2.2 Axioms and expressions

Since the range of possible OWL axiom types and expressions to include in the entailment set is very broad, these will not be represented by a key, but described by listing the axiom pattern or expression pattern instead. For example, the set of entailed atomic subsumptions can be described as 'axioms of the type $A \sqsubseteq B$ for all named, satisfiable classes A, B in $\mathcal{O}$ '. Likewise, we refer to the set of entailed unsatisfiabilities as 'axioms of the type $A \sqsubseteq \bot$ for all named classes A in $\mathcal{O}$ '. This can be easily extended to complex expressions, such as 'axioms of the type $A \sqsubseteq \exists r.B$ for all named, satisfiable classes A, B and named properties r in $\mathcal{O}$ '.

3.2.3 Wanted and unwanted entailments

In the context of debugging an ontology we usually speak of modifying or removing axioms in order to remove *unwanted* entailments from the ontology. One

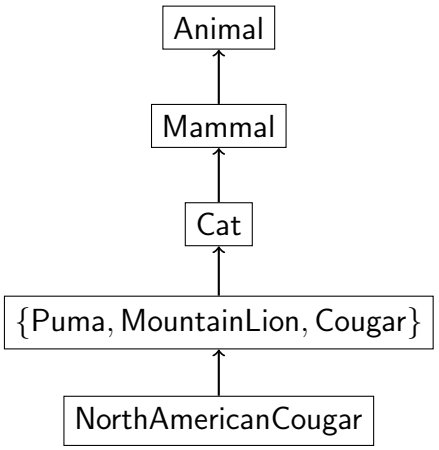
Table 3.1: Entailment set properties and keys

Design decision	Keys	Default
Include / exclude axioms of type $A \sqsubseteq \top$	T^+ / T^-	T^-
Include / exclude asserted axioms	A^+ / A^-	A^-
Include / exclude indirect subsumptions	D^+ / D^-	D^-
Include / exclude non-strict subsumptions	S^+ / S^-	S^-
Representation of equivalent classes		
N-ary equivalent classes axioms	E^n	✓
Exhaustive binary equivalent classes axioms	E^e	–
Select pairwise representatives	E^r	–
Subsumptions between equivalent classes		
Exhaustive subsumption axioms	N^e	✓
Select representative classes	N^r	–
Introduce new name	N^n	–
Import types		
Include / exclude native entailments	I^{n+}/ I^{n-}	I^{n+}
Include / exclude imported entailments	I^{i+} / I^{i-}	I^{i-}
Include / exclude mixed entailments	I^{m+} / I^{m-}	I^{m-}

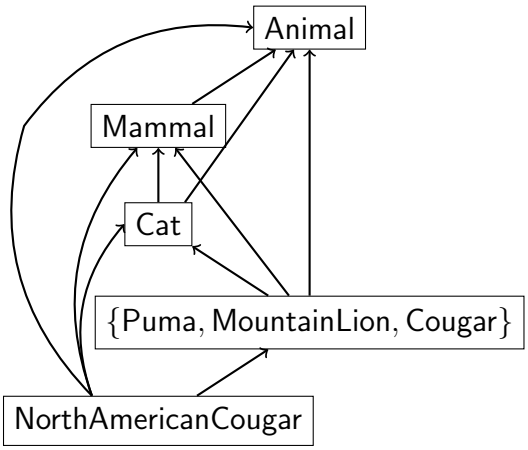
of the main concerns here is to find a modification, i.e. a repair, such that all unwanted entailments are removed, but at the same time as few *wanted* entailments as possible are lost from the ontology. In the most extreme case, a user could simply remove all axioms from an ontology, which would obviously have the desired effect of removing the unwanted entailments, but at the same time this would also remove any relevant information from the ontology. Borrowing some notation from existing work on revising knowledge bases [SFRF12, NRG12], we introduce names for the sets of wanted and unwanted entailments of an ontology: Given a finite entailment set $\varepsilon_{\mathcal{O}} = \varepsilon^+_{\mathcal{O}} \cup \varepsilon^-_{\mathcal{O}}$

- $\varepsilon^+_{\mathcal{O}}$ is the set of *wanted* entailments
- $\varepsilon^-_{\mathcal{O}}$ is the set of *unwanted* entailments.

The decision as to which entailments are unwanted lies with the user; a common choice for $\varepsilon^-_{\mathcal{O}}$ is the set of entailed subsumption axioms of the type $A \sqsubseteq \bot$ for named classes A of an ontology. Given these two sets, we can make a clear distinction between modifications which lead to a positive effect (the removal of entailments in $\varepsilon^-_{\mathcal{O}}$) and those with a negative effect (the loss of entailments in $\varepsilon^+_{\mathcal{O}}$).



(c) Asserted class graph.



(d) Inferred class graph.

Finding a balance between such positive and negative effects is key in the ontology debugging process; a detailed discussion of this issue will follow in Section 4.3.1.

3.2.4 Sample entailment sets

The following examples demonstrate the different criteria and their effects on the size and types of axioms of finite entailment sets for a small ontology.

Example 3.2 (Toy ontology $\mathcal{O}$)**.**

$\mathsf{NorthAmericanCougar} \sqsubseteq \mathsf{Cougar}$	$\mathsf{Mammal} \sqsubseteq \mathsf{Animal}$
$\mathsf{Cougar} \equiv \mathsf{MountainLion}$	$\mathsf{Puma} \equiv \mathsf{Cougar}$
$\mathsf{Puma} \sqsubseteq \mathsf{Cat}$	$\mathsf{Cat} \sqsubseteq \mathsf{Mammal}$

The inferred and asserted class graphs of this ontology are shown in Figures 3.1c and 3.1d, respectively.

Transitive closure The $(AD)^+E^e$ entailment set containing axioms of the type $A \sqsubseteq B$ and $A \equiv B$ for satisfiable classes A, B in $sig(\mathcal{O})$, make explicit the subsumption and equivalence relationships between every single class in the ontology. The set shown in Example 3.3 is the largest finite entailment set to be extracted from the class graph which does not contain any tautologies. The alternative variant $D^+(EN)^e$ of this set excluding asserted subsumption axioms simply discards

the six axioms that occur in the original ontology, yielding a set of 12 axioms.

Example 3.3 ($(AD)^{+}(EN)^{e}$ entailment set of $\mathcal{O}$, 18 axioms)**.**

Cougar ≡ MountainLion	Cougar ≡ Puma
MountainLion ≡ Puma	Puma ⊑ Cat
Puma ⊑ Mammal	Puma ⊑ Animal
MountainLion ⊑ Cat	MountainLion ⊑ Mammal
MountainLion ⊑ Animal	Cougar ⊑ Cat
Cougar ⊑ Mammal	Cougar ⊑ Animal
NorthAmericanCougar ⊑ Puma	NorthAmericanCougar ⊑ MountainLion
NorthAmericanCougar ⊑ Cougar	NorthAmericanCougar ⊑ Cat
NorthAmericanCougar ⊑ Mammal	NorthAmericanCougar ⊑ Animal

Due to the fast growth in size, such a set of entailments may not be suitable for user-facing applications. For the purpose of computing justifications, however, this set set seems most appropriate, as it guarantees that all subsumption relationships are captured.

Transitive reduct The $A^{+}(EN)^{r}$ entailment set based on the transitive reduct of the class graph uses representative elements from each node as well as a pairwise representation of equivalent classes to produce a *minimal* image of the class hierarchy.

Example 3.4 ($A^{+}(EN)^{r}$ entailment set, 6 axioms)**.**

NorthAmericanCougar ⊑ Cougar	Cougar ≡ MountainLion
MountainLion ≡ Puma	Puma ⊑ Cat
Cat ⊑ Mammal	Mammal ⊑ Animal

This entailment set contains all the information that would be necessary for a user to understand (or be able to infer) the relationships in the ontology. The class Puma was selected to represent the node labelled with the equivalent classes $\{\mathsf{Cougar}, \mathsf{Puma}, \mathsf{MountainLion}\}$, as the resulting axiom $\mathsf{Puma} \sqsubseteq \mathsf{Cat}$ is part of the asserted ontology. That is, in this case, the function $Rep(n)$ is used to return a random class name of the equivalent classes node, giving preference to classes that would allow us to generate a subsumption axiom which is already asserted in $\mathcal{O}$. Defining $Rep(n)$ to select any of the two other classes would have been appropriate, too, but would have resulted in a larger entailment set, namely the axioms shown in 3.4 plus either $\mathsf{Cougar} \sqsubseteq \mathsf{Cat}$ or $\mathsf{MountainLion} \sqsubseteq \mathsf{Cat}$.

3.3 Entailments in OWL applications

A number of OWL applications provide methods for generating 'the inferred' version of an ontology, or offer views of 'selected entailments'. From working with these tools, we have found that the notion of entailments does not have a consistent interpretation across different applications. In this section, we briefly outline some of the examples for the use of entailments in OWL tools as well as analytical applications, and show how they would benefit from a flexible and transparent definition of entailment sets.

3.3.1 Inferred ontology generation in the OWL API

The OWL API provides the Java class `InferredOntologyGenerator`, which allows users to 'fill' a new ontology with the desired type of entailments, such as inferred atomic SubClassOf axioms. By default, this method only retrieves the *direct* named superclasses of a named class when using the OWL API's `InferredSubClassOfAxiomGenerator`; any of the properties discussed in the above sections cannot be specified when generating entailments. By extending the `InferredAxiomGenerator` to accept custom configurations, we can simply take into account the additional design decisions described above. We have implemented such a custom entailment generator on top of the OWL API which uses standard configuration files to determine the exact set of axioms to be generated.[6]

[6] `http://owl.cs.manchester.ac.uk/entailments`

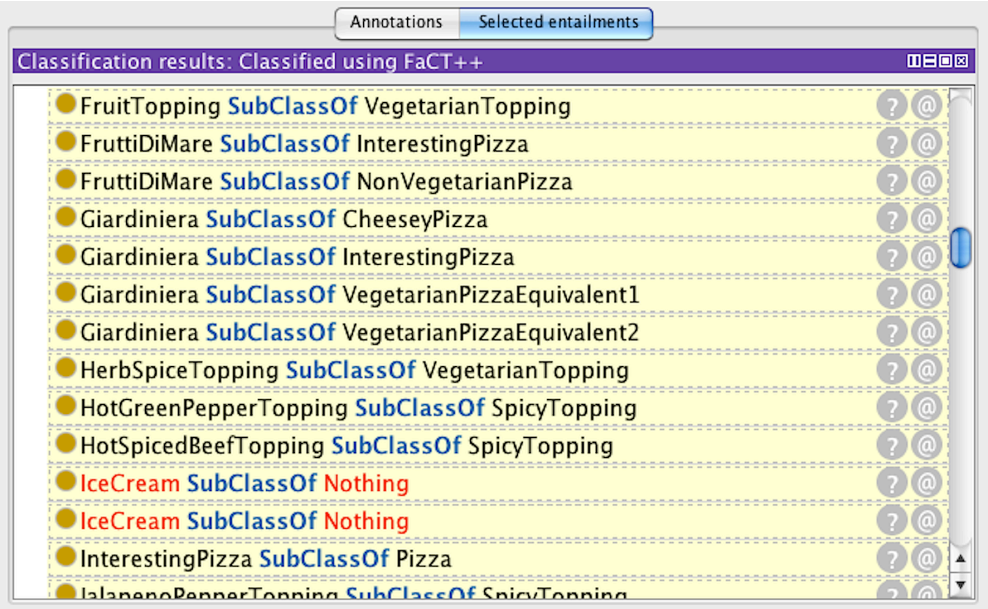


Figure 3.1: Screenshot of the 'Selected Entailments' tab in Protégé 4

3.3.2 Presenting entailments to end-users

The ontology editor Protégé 4 comes with a 'selected entailments' tab which shows a list of atomic SubClassOf, SubPropertyOf, and Type (class assertion) axioms. The tool also offers the option to 'Save inferred axioms as ontology' which saves asserted and inferred axioms as a new OWL ontology. Similarly, Top Braid Composer offers to 'Save [the] inference graph' as a new file. None of the editor offers any further explanation to how these entailments were extracted in the classification process. It may even seem surprising to the end-user that some tautological axioms, such as $A \sqsubseteq \top$, are displayed in Protégé 4's 'selected entailments' panel (shown in Figure 3.1) for *some* classes, while others are missing. The current settings panel for the entailment tabs provides checkboxes for the desired axiom types which are then generated by the corresponding `InferredAxiomGenerator` implementations in the OWL API. This could be simply extended by integrating settings options for the design decisions described above, thus offering a more transparent and flexible view of such 'selected entailments'.

3.3.3 Ontology publishing

Ontologies that are available on the web may be published as 'compiled' versions, which include the ontology and its entailments of some description. The OWL

version of the National Cancer Institute (NCI) Thesaurus, for example, 'includes inferred relationships'.[7] There is, however, no definition of what is regarded as an inferred relationship, how these relationships are determined, and what the selection criteria is. This may leave users wondering what kinds of information they are dealing with, and what implications this has for their understanding of the ontology.

3.3.4 Metrics and analytical applications

Analytical applications that consider the number and types of entailments in order to infer ontology metrics depend on clearly defined finite entailment sets as the basis of their measurements, i.e. *what exactly* is measured. The entailment set used to generate metrics must be well defined and independent from a particular implementation or *individual modifications* of results provided by the OWL API. From anecdotal evidence we know that developers of analytical tools frequently modify the results of the OWL API's `InferredAxiomGenerator` for use in their application, for example by removing entailments of the type $A \sqsubseteq \top$.

One such application is the use of entailment and justification metrics for the purpose of identifying ontology design styles [MSR11] which ensures transparent and consistent measurements across different applications by clearly specifying the type of entailment set to be extracted.

Another example is the counting of entailments which are lost in the debugging process of the OWL SafeMode tool [Sch11] in order to evaluate the quality of a repair step, which requires a stable way of counting entailments that is not sensitive to the irregularities discussed above.

Finally, some approaches to ontology *diffs*, that is, determining and quantifying the semantic and syntactic differences between two ontologies (e.g. two versions of an ontology), make heavy use of finite entailment sets. In these approaches the difference between two ontologies is measured based on the (countable) differences in their (finite) entailment sets. Gonçalves et al. [GPS11b, GPS12a] present several approximations to diffs which are based on increasingly complex types of entailment sets, such as entailed atomic subsumptions, subsumptions between complex class expressions already found in the ontologies, and a *grammar diff* which generates new complex expressions to check for entailment.

[7] `http://evs.nci.nih.gov/ftp1/NCI_Thesaurus/ReadMe.txt`

3.3.5 Imported and native entailments in BioPortal

In a survey of 42 OWL and OBO ontologies from the NCBO BioPortal [BHPS11], we found that imported entailments (in the $(AI^{nim})^+$ set of atomic subsumptions of type $A \sqsubseteq B$ for satisfiable classes A, B) did in fact have a visible effect on the number of entailments found in those ontologies. In this corpus, the upper-level ontology *Basic Formal Ontology (BFO)* [GSG04] contributed significantly to the skewing of entailment counts. Table 3.2 shows an overview of the ontologies in the survey.

Table 3.2: Ontologies and imported entailments in the NCBO BioPortal.

	Importing BFO			Other imports		
Ontologies	Total	Imported	Mixed	Total	Split	No imports
42	7	3	4	7	5	28

7 ontologies in the sample corpus imported BFO, of which 3 had no native entailments at all, but only 70 imported entailments which all originated from BFO. A further 4 ontologies had 70 imported entailments from BFO, plus additional entailments which were either native or imported from ontologies other than BFO.

Additionally, a further 7 ontologies had imported entailments from ontologies other than BFO; for 5 of these, however, the imports could be attributed to the ontology being intentionally split up over several files with similar file names. Examples of these included the *Chemical Information* ontology [KDLD08], which had 1 native entailment, 72 imported entailments from an ontology titled 'cheminf-external', and 3 entailments from 'cheminf-core', or the *Semanticscience Integrated Ontology* (SIO),[8] which had imported entailments from an ontology called *sio-core.owl*.

Finally, 28 ontologies did not have any imported entailments at all, which could be either due to them having no imports, the imported ontology having no entailments that matched our criteria, or missing imports, which were ignored in the pre-processing stage when downloading the ontologies from BioPortal.

The example of the BioPortal ontologies shows how an analysis or comparison of ontologies based on the number of entailments and justifications needs to

[8] `http://code.google.com/p/semanticscience/wiki/SIO`

pay attention to their import structure; in the case of the BFO imports, the 3 ontologies which had only entailments from BFO would have appeared to have exactly the same number of entailments and justifications, and thus might have been wrongfully classified as similar in terms of their expressivity and inferential power.

3.4 Summary and conclusions

In this chapter, we presented a discussion of the issues surrounding finite entailments sets for OWL ontologies. While the classification and computation of inferences is a standard reasoning task in the ontology engineering process, there exist ambiguities in ontology tools as to what constitutes the 'set of entailments' of an ontology. We highlighted various design decisions to be made in order to clearly specify a non-ambiguous finite entailment set of an OWL ontology: how to deal with tautologies, whether to include asserted axioms, how to deal with transitive and non-strict subsumptions, and how to make the transition from nodes and edges in an abstract class graph to concrete sets of *axioms* which are presented to a user or used in an OWL application. We also highlighted the problem of *imported entailments* and how these can skew ontology metrics based on entailments, which was demonstrated using examples from a corpus of biomedical ontologies. The examples found in this chapter illustrate the wide scope of different entailment sets that can be specified depending on the application, ranging from minimal representations which are suitable for user-facing applications, to exhaustive, yet finite, sets which may be used for analytical purposes.

The outcome of the work presented in this chapter is a deeper insight into what we mean by the 'set of entailments' of an ontology, alongside a set of properties which can be used to precisely define a finite entailment set that is fit for a specific purpose. The main benefit of this set of properties is that it is defined from an *application* perspective, which makes the effects of individual choices, such as whether to include indirect subsumptions, easier to understand for a user. Finally, we have made available a prototypical entailment generator on top of the OWL API, which allows the straightforward specification of finite entailment sets using a custom configuration file. The understanding and tools gained here lay the foundations for our discussion of the relations between justifications for entailment sets, which will be presented in the following two chapters.

Chapter 4

The justificatory structure of OWL ontologies

In current ontology explanation tools, justifications are generally represented as individual entities in relation to a single entailment. However, this view neglects the rich relations that can be found between *multiple* justifications for both single and multiple entailments of OWL ontologies. Previous surveys of OWL ontologies have shown that the majority of ontologies used in practice contain some entailment which has more than one justification, with many ontologies generating a few dozen and up to several hundred justifications for a single entailment [BPS10b, Hor11a, BHPS11].

Several properties of justifications, such as the number of justifications for an entailment, the size of justifications in an ontology, to which extent the ontology contains entailments which are entailed only by themselves, and which proportion of the ontology participates in justifications, all give us an insight into structural relationships in the ontology beyond standard metrics; we can say that the justifications make the *implicit* relationships between the entitites and axioms of an ontology *explicit*.

Further, justifications for a single or multiple entailments are not disjoint subsets of an ontology, but they often *overlap* to a certain extent, sharing one or more axioms between them. This is relevant for both understanding as well as repairing the entailments of such justifications: shared axioms may[1] lead to a smaller hitting set, and therefore to a smaller repair for a set of entailments. Further, shared axioms may indicate common lemmas [HPS09], i.e. intermediate entailments, which can assist users in understanding (parts of) multiple justifications at the same time rather than treating each one independently, thus reducing the effort required to deal with multiple justifications.

[1]It is important to point out that a shared axiom does *not* necessarily have to be an incorrect axiom; therefore, it is not guaranteed to be part of the hitting set that leads to a repair.

In this chapter, we will introduce the notion of *justificatory structure* of an OWL ontology. First, we present a categorisation of entailments based on the types of justifications they have in an ontology. We then lay out the motivation for analysing the relations between justifications in an ontology and define various structural aspects which may be of interest for ontology development and analysis. We introduce a graph-based representation of the justifications and entailments in an ontology and describe the generation and implementation of such a *j-graph*. Along with Chapter 5, the work presented in this chapter constitutes the core of our research into the justificatory structure of OWL ontologies.

4.1 Categories of justifications and entailments

Following on from our discussion of different entailment sets in the previous chapter, we will now take a closer look at the different types and scenarios of justifications we encounter when analysing a finite entailment set of an ontology. While it is certainly the case that justifications are essentially equal—they are minimal entailing axiom subsets—we may treat them differently depending on the axioms they contain, and depending on their relation to the entailment in question.

4.1.1 Self-justifications and self-supporting entailments

Any justification which is simply the entailed axiom itself is classified as a *self-justification*:

Definition 4.1 (Self-justification)**.** *A justification $\mathcal{J}$ for an entailment η is a* self-justification *if $\mathcal{J} = \{\eta\}$.*

We distinguish between entailments which have a self-justification in addition to other, more complex, justifications, and entailments which have *only* a self-justification and no other justifications. Referring back to our discussion of finite entailment sets in Chapter 3, these entailments are commonly referred to as 'asserted, but not inferred'; however, in order to avoid ambiguity caused by the word *inferred* (as an asserted axiom can also be inferred from an ontology), we denote such as *self-supporting* entailments:

Definition 4.2 (Self-supporting entailment)**.** *An entailment η is* self-supporting *if $Justs(\eta) = \{\{\eta\}\}$.*

There are different reasons why an entailment might have a self-justification *in addition* to other justifications:

1. The entailment was not asserted in the ontology to begin with, but explicitly added after it was found to be inferred. This could be in order to improve reasoner performance, to make the entailment visible to the *developer* without using a reasoner (e.g. if the classification time is very high), or to make the inferences visible to *end-users* in an ontology browser interface which does not support reasoning.
2. The ontology modeller does not use a reasoner during the engineering process, thus is not aware that the subsumption is already entailed by the ontology, which means they might explicitly assert information which already exists implicitly.
3. The additional justifications are a side-effect of other axioms that were added to the ontology without the aim of causing the entailment, or with the aim of causing this *and* other entailments.

Without additional information, such as axiom annotations or change logs of an ontology, it is not possible to tell the intentions of an ontology developer when adding an axiom which causes additional justifications or self-justifications. Moreover, developers may not even be *aware* of the full impact that a modification has on the ontology. We therefore treat self-justifications and additional justifications on a purely logical level, disregarding the reasons for those multiple justifications.

4.1.2 Atomic subsumption chains

An *atomic subsumption chain justification* in an ontology $\mathcal{O}$ is a set of axioms of the type $A_1 \sqsubseteq A_2$, $A_2 \sqsubseteq A_3$, $\ldots$, $A_{n-1} \sqsubseteq A_n$ for an entailment $A_1 \sqsubseteq A_n$ where A_i is a named class in $sig(\mathcal{O})$. It is clear to see that such an atomic subsumption chain justification of two or more axioms always represents an *indirect* subsumption, which we have discussed in Chapter 3. Take, for example, the following ontology:

Example 4.1.

$$\mathcal{O} = \{A \sqsubseteq B, B \sqsubseteq C, A \sqsubseteq C\}$$

The A^+D^- entailment set of atomic subsumptions is the set $\{A \sqsubseteq B, B \sqsubseteq C\}$, whereas the $(AD)^+$ entailment set is simply $\mathcal{O}$ itself. While each entailment in

$(AD)^+$ has a self-justification, the entailment $A \sqsubseteq C$ has an additional atomic subsumption chain justification $\mathcal{J} = \{A \sqsubseteq B, B \sqsubseteq C\}$. Despite $\mathcal{J}$ being slightly less trivial than a self-justification, it is still simply based on the transitivity of subsumption. This implies that the interactions between axioms in the ontology (with respect to the class hierarchy) are trivial enough to be inferred by a structural reasoner.

Note that it is certainly possible for an entailment to have multiple atomic subsumption chain justifications. For example, adding the axioms $\{A \sqsubseteq E, E \sqsubseteq C\}$ to the above example would add another atomic subsumption chain justification for the entailment $A \sqsubseteq C$.

In our survey of the justificatory structure of OWL ontologies presented in Chapter 7, we find that many ontologies which only contain self-justifications for direct subsumptions often have only atomic subsumption chain justifications for indirect subsumptions; that is, the information content of the ontology does not reach beyond its asserted class graph.

4.1.3 Complex justifications

Finally, we consider all justifications that are not self-justifications or atomic subsumption chains to be *complex* justifications as opposed to *trivial* self-justifications or atomic subsumption chain justifications. It is obvious that the level of complexity can vary strongly between justifications, depending on their size and the constructors used in their axioms. However, for the purpose of categorising justifications, such a coarse distinction between self-justifications, atomic subsumption chains, and *other* justifications suffices.

It is clear to see that entailments can have both trivial and complex justifications. Assume we add further axioms to the above example ontology $\mathcal{O}$ to obtain $\mathcal{O}' = \mathcal{O} \cup \{A \sqsubseteq \exists r.D, \exists r.D \sqsubseteq C\}$. The entailment $A \sqsubseteq C$ will then have another, complex, justification $\mathcal{J} = \{A \sqsubseteq \exists r.D, \exists r.D \sqsubseteq C\}$ in addition to its self-justification and atomic subsumption chain justification.

4.1.4 Categorising entailments and ontologies

If we consider the categorisation of justifications from the point of view of their corresponding entailments, we can also arrange entailments and ontologies into a hierarchy based on their justification. This will be of use when analysing the

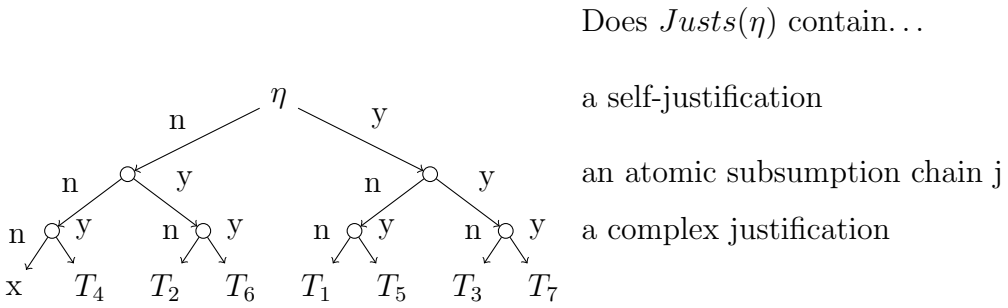


Figure 4.1: A decision tree for categorising entailments.

justificatory structure of an ontology, as it allows us to decide which entailments and justifications to include in specific metrics. For example, when counting the numbers of justifications per entailment, we may want to treat trivial and non-trivial justifications separately in order to ensure that we are comparing equally relevant justifications.

Figure 4.1 shows a decision tree for categorising an entailment based on its justifications. Given an entailment, the decision tree accepts a simple yes/no in the presence/absence of each type of justification, leading the entailment to be sorted into one of seven categories. We have labelled these categories with types T_1 through T_7 based on the number obtained if we regard the respective path in the decision tree as a (reversed) binary number. This results in uneven numbers for entailments which have self-justifications, and even numbers for those entailments that have only non self-justifications.

This categorisation process can be applied to ontologies in order to rank them based on the justification types they contain (given some finite entailment set). This is of use when analysing the justificatory structure of an ontology, as it allows us to separate relevant ontologies (those that do contain entailments with complex justifications) from irrelevant ones (those that only contain trivial entailments).

4.2 Representing justifications as j-graphs

In order to talk about the justificatory structure of an ontology, we require some way of representing structural aspects in an accessible way. The set of entailments, their justifications, and the axioms occurring in these justification can be easily represented as a graph, which allows us to describe aspects of justificatory structure using standard graph metrics such as the in- and out-degrees of nodes.

In this section, we first define the terminology for different aspects of entailment and justification sets, which then allows us to define the *justification graph* for a given set of entailments.

4.2.1 J-graph definition

Before defining j-graphs, we need to introduce the building blocks of these graph representations of justifications. Recall the notion of $Justs(\varepsilon_{\mathcal{O}})$ which refers to the set of all justifications for the entailments in a particular finite entailment set of an ontology $\mathcal{O}$. Further, we define the set of all axioms occurring in all justifications for a particular entailment set:

Definition 4.3 (Justification axioms)**.**

$$JustAx(\varepsilon_{\mathcal{O}}) = \{\alpha \mid \textit{there is a } \mathcal{J} \in Justs(\varepsilon_{\mathcal{O}}) \textit{ s.t. } \alpha \in \mathcal{J}\}$$

Based on our definitions of justification sets and justification axioms, we can now define the justification graph of an ontology $\mathcal{O}$ with respect to an entailment set $\varepsilon_{\mathcal{O}}$. A justification graph (*j-graph*) $G(\varepsilon_{\mathcal{O}})$ is a directed graph whose set of nodes is the union of the set of axioms $\varepsilon_{\mathcal{O}}$ which are entailed by $\mathcal{O}$ and the set $JustAx(\varepsilon_{\mathcal{O}})$ of axioms that participate in justifications for these entailments, together with the set of all justifications $Justs(\varepsilon_{\mathcal{O}})$. The graph does not contain axioms in $\mathcal{O}$ which are neither in the entailment set, nor in $JustAx(\varepsilon_{\mathcal{O}})$, as these would not have any incoming or outgoing edges, thus not adding any information content.

Definition 4.4 (Justification graph)**.** *Given a set of entailments $\varepsilon_{\mathcal{O}}$, their justifications $Justs(\varepsilon_{\mathcal{O}})$, and justification axioms $JustAx(\varepsilon_{\mathcal{O}})$, a* justification graph *is a graph $G(\varepsilon_{\mathcal{O}}) = (\varepsilon_{\mathcal{O}} \cup Justs(\varepsilon_{\mathcal{O}}) \cup JustAx(\varepsilon_{\mathcal{O}}), E_1 \cup E_2)$ where*

- $E_1 = \{(u, v) \in \varepsilon_{\mathcal{O}} \cup JustAx(\varepsilon_{\mathcal{O}}) \times Justs(\varepsilon_{\mathcal{O}}) \mid u \in v\}$ *and*
- $E_2 = \{(v, w) \in Justs(\varepsilon_{\mathcal{O}}) \times \varepsilon_{\mathcal{O}} \mid v \in Justs(\mathcal{O}, w)\}$.

Figure 4.2 shows a j-graph of the small sample entailment and justification set in Example 4.2. As we can see in the example graph on nodes $a1$, $a10$, $a11$ and $a12$, an axiom node in the graph with an in-degree ≥ 1 corresponds to an entailment in $\varepsilon_{\mathcal{O}}$. An axiom node with an out-degree ≥ 1 correspond to an axiom

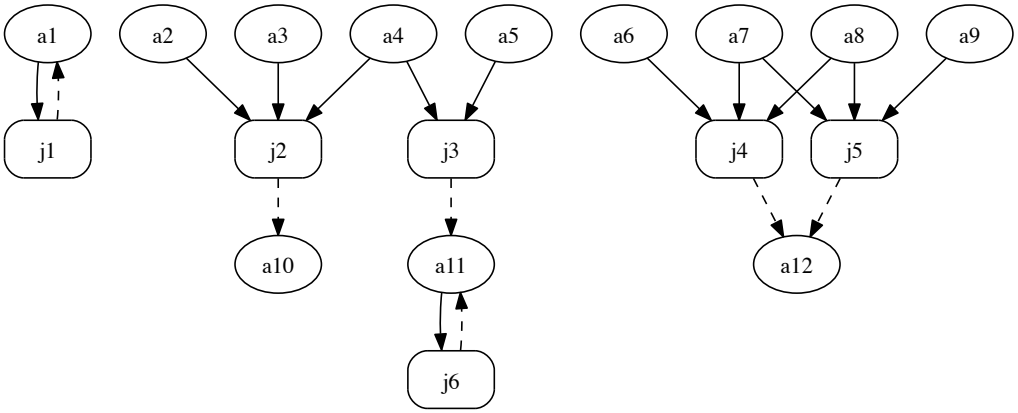


Figure 4.2: An example of a j-graph for justifications and entailments.

in $JustAx(\varepsilon_{\mathcal{O}})$; this node type is represented by nodes $a1$ through $a9$ in the example graph.

Example 4.2.

$$
\begin{aligned}
j1 =& \{a1\} \models a1 \\
j2 =& \{a2, a3, a4\} \models a10 \\
j3 =& \{a4, a5\} \models a11 \\
j4 =& \{a6, a7, a8\} \models a12 \\
j5 =& \{a7, a8, a9\} \models a12 \\
j6 =& \{a11\} \models a11
\end{aligned}
$$

A self-justification corresponds to a cycle between an axiom node and a justification node (which, in turn, only has this one incoming edge), as justifications $j1$ and $j6$ show. A self-supporting entailment ($a1$ in the example graph) is represented by an axiom node which has no other incoming edges in addition to the cycle.

Note that, while some j-graphs may be tripartite, an axiom node can be both in the set $JustAx(\varepsilon_{\mathcal{O}})$ and in $\varepsilon_{\mathcal{O}}$, i.e. have an outgoing edge to a justification node and an incoming edge from a justification node, as we can see from both axioms $a1$ and $a11$ in the example graph. Thus, a j-graph is bipartite, but not

guaranteed to be tripartite.

There exists a *unique* j-graph for every set of entailments, as the set of justifications for an entailment set is unique in an ontology, and the edges in the j-graph follow from these unambiguous relations.

Recall that the number of justifications for an entailment can potentially be exponential in the number of axioms in $\mathcal{O}$. This affects the completeness of a j-graph for a given finite entailment set, as it may not be practical to compute *all* justifications for *all* entailments in the set. Despite the general feasibility of computing justifications, as shown in [Hor11a], computing all justifications for a large finite entailment set can still require more time than an ontology user may consider practical.

This means that we either need to sample a subset of entailments, or only generate a certain number of justifications for each entailment in the given entailment set—or both. While this omission of information is obviously problematic for users attempting to find a repair for *all* justifications for a given entailment set, the only solution to such computational problems is incremental repair. Therefore, we only consider the j-graph for an entailment set to represent the particular set of entailments and justifications that could be processed, even if these are not all that exist in the ontology.

4.2.2 J-graph generation

Construction Generating a j-graph to represent the justificatory structure of an OWL ontology is a fairly straightforward process. First, all axioms in the union of the ontology and the selected entailment set $\varepsilon_{\mathcal{O}}$ are labelled with a unique identifier a_i. We then compute the justifications for the entailments in $\varepsilon_{\mathcal{O}}$; the justifications are also labelled with unique identifiers j_k. The axiom labelling can also be performed on an existing set of justifications, which has no effect on the general structure of the process.

The j-graph is then constructed as follows: generate a node for each axiom in $JustAx(\varepsilon_{\mathcal{O}})$ and $\varepsilon_{\mathcal{O}}$ (each justification in $Justs(\varepsilon_{\mathcal{O}})$) and label it with the respective identifier. For each justification node labelled j_k, generate an edge (a_i, j_k) for each axiom a_i in the justification labelled j_k; this will create the edges representing the relation 'axiom is used in justification'. For each entailment node a_i with a justification j_k, generate an edge (j_k, a_i) which represents the relation 'justification for entailment'. This construction requires only a single pass over

all given justifications, axioms, and entailments.

Implementation The above algorithm has been implemented using the OWL API version 3.2.4, and the JGraphT[2] version 0.8.3 graph library for representing the j-graph. Construction of the graph on a set of existing justifications and entailments is performed quickly, with the average time to construct a graph being less than 10 seconds in a set of ontologies ranging from 2 to approximately 11,000 justifications.

4.3 Justificatory structure

Using the j-graph to describe relations between entailments, their justifications, and the axioms in the justifications, we can now discuss the various facets of the *justificatory structure* of an OWL ontology. In a first instance, the justificatory structure provides additional ontology metrics; beyond these metrics, insights into the structure can support user understanding when coping with multiple justifications and/or multiple entailments. Further, in Chapter 6 we will introduce debugging techniques that are directly based on the interaction with the j-graph for a user-defined entailment set.

Note that in this section we refer to 'the entailments' and 'the justifications' of an ontology, which requires a more precise specification. In what follows, we assume these entailments to be the $(AD)^+$ set of entailments which includes all asserted and inferred native, direct and indirect (non-tautologic) atomic subsumptions of the type $A \sqsubseteq B$ for satisfiable classes A, B which are entailed by $\mathcal{O}$, as this provides a clearly defined and natural starting point for exploring the properties of an ontology.

For ease of reading, we will use the terms *justification*, *axiom*, and *entailment* interchangeably with *justification node*, *axiom node*, and *entailment node* in the remainder of this chapter.

4.3.1 Axiom properties

The axioms occurring in justifications for a given set of entailments can have different properties, which we denote as axiom *frequency*, *impact*, and *semantic*

[2] `http://jgrapht.org/`

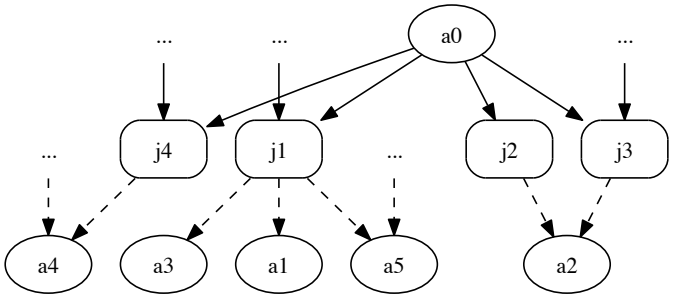


Figure 4.3: J-graph illustrating axiom frequency, impact, and semantic relevance. Axiom $a0$ has a frequency of 4, impact of 5, and semantic relevance of 3.

relevance. These three terms describe similar, but slightly distinct, measures of the effect an axiom in $JustAx(\varepsilon_{\mathcal{O}})$ has on a set of justifications and entailments. Figure 4.3 shows an example j-graph which exhibits different values of frequency, impact, and semantic relevance for the axiom node $a0$. Blank nodes labelled with '...' indicate the existence of some additional axiom(s) occurring in a justification, respectively additional justification(s) for an entailment.

Frequency

The *frequency*, also referred to as *power* or *arity*, of an axiom is the number of justifications (for a given entailment set $\varepsilon_{\mathcal{O}}$) the axiom occurs in. Removing an axiom with frequency n from the ontology will break n justifications for the entailments in $\varepsilon_{\mathcal{O}}$. Given that there can exist multiple justifications for an entailment, the frequency of the axioms in a set of justifications does not necessarily tell us how a removal would affect the actual *entailments* of those justifications. In a j-graph, the frequency of an axiom node corresponds to its out-degree; an axiom with a frequency greater than 1 corresponds directly to a justification overlap of size 1 between the justifications the axiom occurs in:

Definition 4.5 (Frequency)**.** $freq(u) = |\{v \mid (u, v) \in E_1\}|$ *for* $u \in JustAx(\varepsilon_{\mathcal{O}})$.

The axiom node $a0$ in Figure 4.3 has edges to four justifications ($j1$, $j2$, $j3$, $j4$); thus, $a0$ has a frequency of 4.

Impact

The *impact* of an axiom is, in some sense, an extension of the axiom frequency. It refers to the number of *entailments* in $\varepsilon_{\mathcal{O}}$ that are directly affected by the axiom, as the axiom occurs in some justification for the entailments. Removing an axiom with an impact of m from the ontology *may* break m entailments in $\varepsilon_{\mathcal{O}}$, assuming there are no other justifications for those m entailments. Axiom frequency and impact coincide if each justification that the axiom occurs in has exactly one (distinct) entailment, and the impact is *less* than the axiom frequency if the axiom occurs in multiple justifications for the same entailment.

In a j-graph, the impact of an axiom node u is defined as the sum of entailment nodes w that have an incoming edge from a justification node v such that $(u, v) \in E_1$ and $(v, w) \in E_2$:

Definition 4.6 (Impact)**.** $impact(u) = |\{w \mid \exists v, w \ s.t. \ (u, v) \in E_1, (v, w) \in E_2\}|$ *for* $u \in JustAx(\varepsilon_{\mathcal{O}})$.

The axiom node $a0$ in the j-graph in Figure 4.3 has an impact of 5, as it occurs in the justifications ($j1$, $j2$, $j3$, $j4$) for all five distinct entailments in the graph.

In the context of repairing a given set of entailments, we will need to consider the impact of axioms both with regard to the set of *unwanted* as well as the set of *wanted* entailments: removing a high-impact axiom from an ontology may break a large number, or all, unwanted entailments (e.g. unsatisfiable classes), but it may also lead to the removal of relevant information. Referring to our definitions of wanted and unwanted entailment sets in Section 3.2.3, we can distinguish between the *positive* and *negative* impact of an axiom:

Definition 4.7 (Positive and negative impact)**.**

$$impact^{-}(\alpha) = \left|\{\eta_i \mid \eta_i \in \varepsilon^{+}_{\mathcal{O}}, \alpha \in JustAx(\varepsilon^{+}_{\mathcal{O}})\}\right|$$
$$impact^{+}(\alpha) = \left|\{\eta_i \mid \eta_i \in \varepsilon^{-}_{\mathcal{O}}, \alpha \in JustAx(\varepsilon^{-}_{\mathcal{O}})\}\right|$$

In words, the negative impact of an axiom α is the number of *wanted* entailments that are entailed by the justifications α occurs in, and the positive impact is the number of *unwanted* entailments that are entailed by the justifications α occurs in. Note that the plus and minus signs for positive/negative impact and

wanted/unwanted entailment sets are swapped, i.e. $impact^-$ refers to $\varepsilon^+_{\mathcal{O}}$, and vice versa.

Semantic relevance

Finally, the *semantic relevance* of an axiom, as defined by Kalyanpur [Kal06], is the number of entailments that are *dependent* on the axiom. Semantic relevance is closely related to the impact of an axiom, taking into account only those entailments that have no additional justifications (or *only* justifications that contain the axiom). In the context of debugging a set of entailments, the semantic relevance of an axiom is the most meaningful of three axiom measures, as it gives a clear indication of the effect axiom removal will have on a set of entailments. Based on the definition given by Kalyanpur [Kal06] , the semantic relevance of an axiom node u labelled with an axiom α in a j-graph is calculated as follows:

Definition 4.8 (Semantic relevance)**.**

$$SR(u) = |\{w \mid \exists v, w \ s.t.\ (u,v) \in E_1, (v,w) \in E_2 \wedge \nexists v' \ s.t.\ (v',w) \in E_2 \wedge (u,v') \notin E_1\}| \\ for\ u \in JustAx(\varepsilon_{\mathcal{O}}).$$

Axiom $a0$ in the j-graph in Figure 4.3 has a semantic relevance of 3, as it occurs in justifications ($j1$, $j2$, $j3$) for exactly three entailments ($a1$, $a2$, $a3$) which do not have any additional justifications that do not contain $a0$.

Activity

We can define the *activity* of an ontology $\mathcal{O}$ with respect to a finite entailment set $\varepsilon_{\mathcal{O}}$ of $\mathcal{O}$. The activity corresponds to the total number of axioms that occur in justifications which are *not* self-justifications, that is, the size of the subset of the ontology which actively participates in entailment:

Definition 4.9 (Activity)**.**

$$activity(\mathcal{O}, \varepsilon_{\mathcal{O}}) = \frac{\sum |\mathcal{J}_i|}{|\mathcal{O}|} \ where\ \mathcal{J}_i \in Justs(\varepsilon_{\mathcal{O}}),\ \mathcal{J}_i\ is\ not\ a\ self-justification.$$

Likewise, an *active* axiom is an axiom which occurs in a justification that is not a self-justification:

Definition 4.10 (Active axiom)**.** *An axiom* $\alpha \in \mathcal{O}$ *is* active *if* $\alpha \in \mathcal{J}$ *where* $\mathcal{J} \in Justs(\varepsilon_{\mathcal{O}})$, $\mathcal{J}$ *is not a self-justification.*

Note that if we select the entailment set to be the deductive closure of the ontology, then the activity of the ontology is 1, since every axiom participates in the inference.

4.3.2 Properties of justifications

Justificatory redundancy

Following on from the discussion of self-justifications and additional justifications, we can regard the number of justifications per entailment as an indicator of *justificatory redundancy* in the ontology; it demonstrates 'how often the same piece of information is expressed in different ways'. Justificatory redundancy is not to be confused with logical redundancy, as this would imply that it is possible to remove a set of axioms from the ontology without breaking any entailments.

In the j-graph, an axiom node with an in-degree > 1 has redundant justifications; thus the in-degree (average, median, or maximum) of axiom-nodes in a j-graph is a metric for determining the level of justificatory redundancy in an ontology.

Justfication size

The *size* of a justification is the number of axioms it contains; this corresponds to the in-degree of a justification node in the j-graph. Justification size gives us information about the use of entities in an ontology in two ways: first, large justifications can be caused by signatures which spread across multiple axioms, while small justifications indicate that the entities in the ontology are less connected. On the other hand, justifications with small numbers of axioms can also be caused by long axioms, that is, axioms containing many subexpressions, whereas justifications with many axioms may be caused by shorter axioms.

4.3.3 Relations between justifications

Structural regularities

Two types of patterns can be identified in the context of the justificatory structure, namely *structural isomorphism* between isomorphism and *graph surface patterns*.

Justification isomorphism [HBPS11a] describes a situation where the *axioms* within a set of justifications share an identical structure; for example, the two justifications $\mathcal{J}_1 = \{A \sqsubseteq \exists r.B, \exists r.B \sqsubseteq C\} \models A \sqsubseteq C$ and $\mathcal{J}_2 = \{X \sqsubseteq \exists s.Y, \exists s.Y \sqsubseteq Z\} \models X \sqsubseteq Z$ are isomorphic, as they have the same structure and only differ in the class and property names used. Justification isomorphism will be discussed in great detail in Chapter 5; for now we will focus on structural properties of the j-graph and treat justification axioms as 'black boxes'.

A surface pattern is a structural similarity between sets of nodes and edges in the j-graph, such as subgraphs which match based on their node types and the numbers of ingoing and outgoing edges. Surface patterns in the j-graph reveal modelling similarities in the ontology, regardless of whether the axioms and expressions occurring in the pattern also interact in a similar way. Highlighting a pattern of this type may support user understanding of the modelling in the ontology, while it may also be an indicator for isomorphic justifications. In the example j-graph, the justifications $j4$ and $j5$ both have a similar structure (same in and out-degrees, two shared axioms), which may be an indicator for structurally isomorphic justifications.

Graph components

A graph *component* is a subset of nodes and edges which forms a disconnected subgraph. The number of components of a j-graph provides a measure for the disjointness of justifications in the ontology. The disjointness of justifications is thought to affect the justification computation process, which makes use of Reiter's Hitting Set Tree (HST) algorithm [Rei87] for diagnosis. Recall that in the HST, the nodes are labelled with justifications and the paths constitute hitting sets across the justifications in the tree, i.e. minimal repairs for the justifications. Optimisations for the HST algorithm are mainly based on closing a branch in the tree if the path to it is labelled with a superset of an existing path, which is not possible if the justifications are disjoint. This is thought to lead to a rapid

growth of the HST and have significant negative effects on the performance of computing all justifications for an entailment.

Arbitrary justification overlap

We have already touched upon the subject of justification overlap in the discussion of axiom frequency, as an axiom with a frequency greater than one simple corresponds to a single-axiom overlap between the justifications that the axiom occurs in. The concept of analysing frequently occurring *axiom groups* in justifications for the purpose of finding a suitable repair was proposed by Schlobach in the early stages of justification-based explanation research [Sch05a]. Due to a lack of an efficient implementation, however, the experimental analysis of such *maximal arity cores* was restricted to single axioms, and has since not received any significant attention.

In the j-graph, arbitrary overlap between n justifications corresponds to an intersection of the *incoming neighbours* N_{in} of the justification nodes with a cardinality greater than one. An intersection of size one corresponds to a single axiom with a frequency of n, thus, we are mainly interested in those overlaps which contain more than just one axiom.

Note that when we talk about overlaps between justifications there is no *unique* overlap that a justification axiom occurs in. Instead, an axiom can occur in several overlaps of varying sizes between different justification sets. We can say that an overlapping set has two dimensions: its *size*, which is the number of axioms in the overlap, and its *frequency*, which has the same meaning as the frequency of a single axiom, that is, the number of justifications the overlapping axiom set occurs in.

Root and derived justifications

A special case of justification overlap are root and derived justifications, which we discussed in Section 2.3.4. Recall: a justification $\mathcal{J}'$ is derived from a justification $\mathcal{J}$ if $\mathcal{J}$ (the root justification) is a strict subset of $\mathcal{J}'$. Due to the minimality of justifications, root and derived relations can only occur between justifications for *multiple* entailments. In the j-graph, a root and derived relationship between two justification nodes is defined as a subset relation between the sets of incoming neighbours of the two nodes (where $N_{in}(v)$ denotes the set of nodes that have an outgoing edge to the node v):

Definition 4.11 (Root and derived justifications)**.** *A justification v is* derived *from a justification v' if $N_{in}(v) \subset N_{in}(v')$, where $N_{in}(v) = \{u_i\}$ s.t. $(u_i, v) \in E_1$. A justification which is not derived from any other justification is a* root *justification.*

Note that root and derived justifications were initially introduced as root and derived unsatisfiable classes. However, as the concept holds for arbitrary entailments, we can speak of root and derived entailments in the same way: a root unsatisfiable class is a class which has only root justifications, and likewise, an arbitrary entailment is a root entailment if it has only root justifications.

Example 4.3.

$$\mathcal{O} = \{A \sqsubseteq \exists r.B, \exists r.B \sqsubseteq C, B \sqsubseteq D, \exists r.D \sqsubseteq E\}$$

In Example 4.3, the set comprising the first two axioms is a justification $\mathcal{J}_1$ for $A \sqsubseteq C$, while the set of all four axioms is a justification $\mathcal{J}_2$ for $A \sqsubseteq E$. Here, $\mathcal{J}_1$ is a root justification, and $\mathcal{J}_2$ is derived from $\mathcal{J}_1$. The two entailments are clearly related via the two axioms which constitute the root justification; we can say that $\mathcal{J}_1$ is the *root* of the entailments, while the remaining two axioms act as a *bridge* between them.

When confronted with a situation such as the justifications in Example 4.3, addressing the derived justification first may lead a user to modify or remove the bridging axioms before the root axioms, thus only affecting the derived entailment. This would require the user to repair the root justification in another step, which would double the effort involved to repair both entailments. This example shows that when attempting to repair multiple entailments, addressing root justifications before derived justifications generally requires the inspection of fewer justifications, thus reducing user effort.

Equality and inferential power

Equality is another special case of justification overlap, where the same subset $\mathcal{J}$ of axioms in an ontology $\mathcal{O}$ is a minimal entailing set for several different entailments. We refer to the number of entailments for which $\mathcal{J}$ is a justification as the *inferential power* of $\mathcal{J}$, answering the question 'how much can be expressed with

how little?'. It is clear to see that for non-laconic justifications the inferential power depends largely on the size and complexity of the axioms in the justifications; we can easily construct an axiom with high inferential power by creating a large conjunction of subexpressions, where only some of the subexpressions are relevant for a given entailment. On the other hand, this is not the case for laconic justifications, as these will not contain any superfluous information with respect to an entailment.

For justification equality, the j-graph representation has clear benefits over a representation of multiple justifications as lists of axiom sets. Several justifications for multiple entailments which are equivalent are simply represented by a single justification node; the inferential power of a justification corresponds to its out-degree in the j-graph.

4.4 Summary and conclusions

This chapter introduced the notion of the justificatory structure of OWL ontologies, defined a graph-based representation of the justificatory structure, and gave a brief overview of the implementation of a generator for such graphs. We discussed a strategy for categorising justifications and entailments based on their relevance, the occurrence of multiple justifications in OWL ontologies, and the various properties associated with entailments and their justifications. Some of these measurements, such as the numbers and sizes of justifications for a finite entailment set of an ontology, or the activity of an ontology, that is, the proportion of ontology axioms that occur in justifications for an entailment set, give us insight into the implicit structure of an ontology, thus extending the standard metrics used to describe OWL ontologies.

Other structural aspects are based on relations between multiple justifications, such as shared axioms between justifications. Justifications for both single and multiple entailments may share some axioms, i.e. the justifications *overlap* to a certain extent. Justification overlaps are key in finding good repairs for a set of justifications, as they may lead to a smaller repair (i.e. hitting set) over the justifications. Furthermore, overlapping justification subsets may also support users in *understanding* multiple justifications by generating lemmas, that is, intermediate entailments, which summarise a recurring set of axioms. In the case of multiple entailments, relations such as root and derived and justification equivalence even

seem crucial to reducing user effort for justification understanding and repair, as they highlight the actual (number of) reasons for the entailments to hold. An in-depth discussion of how justificatory structure can be exploited in the debugging process will follow in Chapter 6; for now, we will continue our exploration of justificatory structure by focussing on one particular structural property, namely equivalence relations over justifications in the form of different isomorphisms.

Chapter 5

Justification isomorphism

In the previous chapter, we introduced the justificatory structure of ontologies and treated the axioms as anonymous 'building blocks' of justifications, without paying consideration to the subexpressions occurring in them. In this chapter, we will go one step further and take into account the logical form of the *axioms* occurring in justifications.

Given the high expressivity of modern ontology languages, such as OWL, there is the possibility for great diversity in the logical content of ontologies. The fact that many naturally occurring entailments have multiple justifications indicates that ontologies often overdetermine their consequences. However, the multiplicity of justifications might be due mostly to diverse *material*, not *formal*, grounds for an entailment. That is, the logical form of these multiple reasons could be less diverse than their numbers suggest. Such logical similarity indicates that it is possible group justifications into sets of structurally similar ones by the means of an equivalence relation.

A well-known syntactical equivalence relation in OWL is *structural equivalence*. The OWL Structural Specification[1] states the condition for two OWL objects (named classes, properties, or instances, complex expressions, or OWL axioms) to be structurally equivalent: in short, it defines the objects to be equivalent if they contain the same names and constructors, regardless of ordering and repetition (in an unordered association). The OWL API implements this notion of structural equivalence by default.

A second equivalence relation is *justification isomorphism*[2] which was first introduced in a study of the cognitive complexity of justifications [HBPS11a]: two justifications are isomorphic if they are structurally identical,[3] i.e. the axioms

[1] `http://www.w3.org/TR/owl2-syntax`

[2] Note that, in the spirit of consistent naming, we will use the term 'isomorphism' for the newly introduced equivalent relations despite their not being isomorphisms (i.e. bijective mappings) in the true sense.

[3] Modulo structural equivalence.

contain the same *types of* class expressions and only differ in the class, property and instance names. The following example shows two justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ which we consider to be isomorphic:

Example 5.1 (Isomorphic justifications)**.**

$$\mathcal{J}_1 = \{A \sqsubseteq B \sqcap \exists r.C, B \sqcap \exists r.C \sqsubseteq D\} \models A \sqsubseteq D$$
$$\mathcal{J}_2 = \{E \sqsubseteq B \sqcap \exists s.F, B \sqcap \exists s.F \sqsubseteq D\} \models E \sqsubseteq D$$

It is straightforward to see that in the above justifications the class A in $\mathcal{J}_1$ can be mapped to E in $\mathcal{J}_2$, the property r to s, and the class C to F in order to obtain identical justifications.

While justification isomorphism helps to eliminate the effects of diverging class, property, and instance names, we can also identify types of justifications which may be considered to be very similar despite their use of different constructors:

Example 5.2.

$$\mathcal{J}_1 = \{A \sqsubseteq B \sqcap C, B \sqcap C \sqsubseteq D\} \models A \sqsubseteq D$$
$$\mathcal{J}_2 = \{A \sqsubseteq \exists r.C, \exists r.C \sqsubseteq D\} \models A \sqsubseteq D$$

In this example, the semantics of the complex class expressions $B \sqcap C$ in $\mathcal{J}_1$ and $\exists r.C$ in $\mathcal{J}_2$ are not relevant for the respective entailment; their occurrences in the justifications and their entailments can be replaced by freshly generated class names without affecting the entailment relation. Such a substitution, in turn, would make the two justifications isomorphic.

Likewise, justifications of different lengths may be considered similar if their general structure of reasoning is identical:

Example 5.3.

$$\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq C\} \models A \sqsubseteq C$$
$$\mathcal{J}_2 = \{A \sqsubseteq B, B \sqsubseteq C, C \sqsubseteq D\} \models A \sqsubseteq D$$

These two justifications clearly require the same form of reasoning from a user which means we may consider them to be structurally similar in some way. Strict isomorphism only applies to justifications which contain the same number

of axioms; it does not cover situations like one shown above. However, for the purpose of structuring sets of justifications and analysing the logical diversity of a corpus of justifications, capturing those kinds of similarities illustrated in the above examples would be highly desirable.

In this chapter, we introduce two new equivalence relations, *subexpression-isomorphism* $\approx_s$ and *lemma-isomorphism* $\approx_\ell$, between justifications based on the subexpressions of their axioms and axiom subsets, respectively, and show how these equivalence relations can be used to determine the logical diversity of a set of justifications. We present an algorithm to detect such equivalence relations and discuss their practical implementation as part of the framework for analysing the justificatory structure of OWL ontologies.

5.1 Isomorphism

Justification isomorphism was first introduced as a way to reduce sampling bias in a user study to validate a model for the cognitive complexity of OWL justifications [HBPS11a]. When studying the cognitive complexity of justifications, we can assume that ontology users will perceive isomorphic justifications as equally difficult to understand, taking into account some variation caused by the complexity of class names and domain knowledge. The ontology corpus from which the justifications in the complexity study [HBPS11a] were sampled was reduced from 64,800 to 11,600 non-isomorphic justifications, thus eliminating the risk of sampling bias caused by presenting users a series of structurally identical justifications. Recall that a justification $\mathcal{J}$ is always defined with respect to a particular entailment η; thus, we will use the notation $(\mathcal{J}, \eta)$ to refer to a justification-entailment pair. Justification isomorphism is defined as follows:

Definition 5.1 (Justification isomorphism)**.** *Two justifications $(\mathcal{J}_1, \eta_1)$, $(\mathcal{J}_2, \eta_2)$ are* isomorphic *($(\mathcal{J}_1, \eta_1) \approx_i (\mathcal{J}_1, \eta_1)$) if there exists a bijective renaming ϕ which maps class, property, and instance names in $\mathcal{J}_1$ and η_1 to class, property, and instance names in $\mathcal{J}_2$ and η_2, respectively, such that $\phi(\mathcal{J}_1) = \mathcal{J}_2$ and $\phi(\eta_1) = \eta_2$.*

Remarks

1. The relation $\approx_i$ is symmetric, reflexive and transitive, from which it follows that $\approx_i$ is an equivalence relation and thus partitions a set of justifications. Proofs are omitted as these properties are straightforward to see.

2. Justification isomorphism preserves the numbers and types of axioms and subexpressions in the justifications:
   (a) If $\mathcal{J}_1 \approx_i \mathcal{J}_2$ then $|\mathcal{J}_1| = |\mathcal{J}_2|$.
   (b) Since the mapping ϕ is bijective, we also have $\mathcal{J}_1 \approx_i \mathcal{J}_2$ implies that $|sig(\mathcal{J}_1)| = |sig(\mathcal{J}_2)|$.
   (c) The sets of logical constructs used in $\mathcal{J}_1$ and $\mathcal{J}_2$ coincide.

5.2 Subexpression-isomorphism

From the above definition of isomorphism it follows that only justifications which have the same number and types of axioms and subexpressions can be isomorphic. It is easy to see, however, that justifications can have a similar structure despite their use of different class expressions, as demonstrated in Example 5.2. Furthermore, obfuscating complex expressions which can be substituted by propositional variables reduces superfluous clutter in a justification, thus improving the readability of a justification akin to laconic justifications. This motivates a notion of *subexpression-isomorphism*, an equivalence relation which allows not only the mapping of class names, but also that of complex subexpressions.

The idea of finding similarities between concepts in description logics has been widely explored in the work on *unification* and *matching*, e.g. [BN98, BKBM99, BM09], for the purpose of detecting redundant class descriptions in ontologies. Given two class expressions C and D, a *unification problem* $C \equiv^? D$ asks whether there exists a *substitution* σ, that is, a mapping from a set of named classes in C into the set of class expressions in D, such that $\sigma(C) \equiv \sigma(D)$. The aim of unification is to find a suitable substitution σ which maps atomic classes C to (possibly non-atomic) classes in D such that the two classes are made equivalent.

While the basic idea behind subexpression-isomorphism is based on unification and matching, the concepts are not directly applicable to the given problem of matching justifications. In our case, the inputs are of different shape from the matching problem: the goal is to unify two sets of axioms and the corresponding entailments, rather than matching a single class expression (a class *pattern*) which contains variables to a class expression. That is, we attempt to identify pairs of possibly complex class expressions that, when replaced with names, result in isomorphic justifications. This means that the substitution mechanism as used in unification is not applicable.

We therefore introduce a *justification template* Θ, which functions as the *unifying justification* for the isomorphic justifications and *two* substitutions ϕ_1, ϕ_2 for use in subexpression-isomorphism. We will use freshly introduced entity names x_i for the named classes, properties, and instances in a template Θ. In order for subexpression-isomorphism to be transitive, the justifications will have to fulfil certain side-conditions, which will be used in the proof of Proposition 5.2:

S1 For any C in the range of $\phi 1$ ($\phi 2$), C is not equivalent to $\top$.

S2 For any C in the range of $\phi 1$ ($\phi 2$) is satisfiable in a *single* element of some interpretation, that is, there is some I such that $\left|C^I\right| = 1$.

S3 For any C_1, C_2 in the range of ϕ_1^{ac} (ϕ_2^{ac}, respectively), it must hold that $sig(C_1) \cap sig(C_2) = \emptyset$ and $sig(C_i) \cap sig(\Theta) = \emptyset$, that is, the expressions in the domain and the range of the mappings must be pairwise signature disjoint.

Definition 5.2 (Subexpression-isomorphism)**.** *Two justifications $(\mathcal{J}_1, \eta_1)$, $(\mathcal{J}_2, \eta_2)$ are* s-isomorphic *($(\mathcal{J}_1, \eta_1) \approx_s (\mathcal{J}_2, \eta_2)$) if there exists a justification* template *(Θ, η) and two injective substitutions ϕ_1, ϕ_2 satisfying **S1** to **S3**, such that*

1. $\Theta \models \eta$
2. $\phi_1(\Theta) = \mathcal{J}_1$ *and* $\phi_2(\Theta) = \mathcal{J}_2$
3. $\phi_1(\eta) = \eta_1$ *and* $\phi_2(\eta) = \eta_2$.

The mappings ϕ_1 and ϕ_2 map class, property, and instance names in the template (Θ, η) to subexpressions of $(\mathcal{J}_1, \eta_1)$ and $(\mathcal{J}_2, \eta_2)$, respectively.

Note that in order to be s-isomorphic, the justifications may differ in the number of subexpressions. As with strict isomorphism, however, they must have the same number of axioms: $\mathcal{J}_1 \approx_s \mathcal{J}_2 \rightarrow |\mathcal{J}_1| = |\mathcal{J}_2|$.

Proposition 5.1. *S-isomorphism is a more general case of strict isomorphism: $\mathcal{J}_1 \approx_i \mathcal{J}_2 \rightarrow \mathcal{J}_1 \approx_s \mathcal{J}_2$.*

Proof. Let $(\mathcal{J}_1, \eta_1) \approx_i (\mathcal{J}_2, \eta_2)$ via a mapping ϕ. Then we can consider a template Θ to correspond to $\mathcal{J}_1$, the mapping ϕ_1 is the identity relation *id* which maps $\mathcal{J}_1$ to itself, and ϕ_2 corresponds to the mapping ϕ used for strict isomorphism. Thus $\mathcal{J}_1 \approx_i \mathcal{J}_2$ then $\mathcal{J}_1 \approx_s \mathcal{J}_2$. □

Proposition 5.2. *The relation $\approx_s$ is reflexive, transitive and symmetric for description logics without nominals up to $\mathcal{SRIQ}$.*

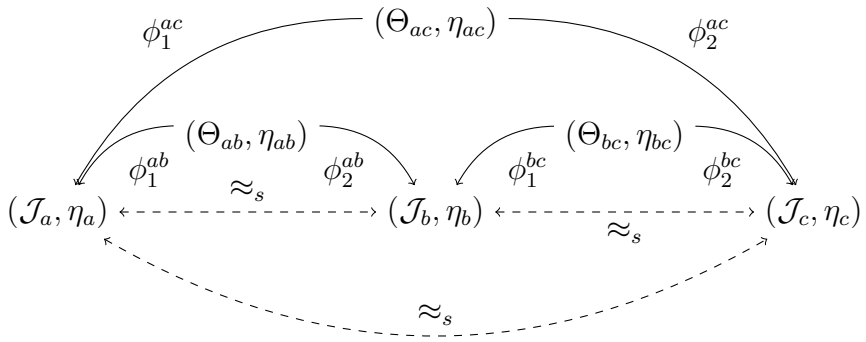


Figure 5.1: Graph representing the relations between three justifications which are subexpression-isomorphic via transitivity.

Proof. Reflexivity and symmetry of the relation are straightforward to see; therefore we will only prove the transitivity of subexpression-isomorphism for logics not containing nominals.

Let $(\mathcal{J}_a, \eta_a) \approx_s (\mathcal{J}_b, \eta_b)$ via $\phi_1^{ab}, \phi_2^{ab}, (\Theta_{ab}, \eta_{ab})$ and $(\mathcal{J}_b, \eta_b) \approx_s (\mathcal{J}_c, \eta_c)$ via $\phi_1^{bc}, \phi_2^{bc}, (\Theta_{bc}, \eta_{bc})$, respectively. We aim to show that given this case it is always possible to construct two mappings ϕ_1^{ac}, ϕ_2^{ac} and a template (Θ_{ac}, η_{ac}), such that $(\mathcal{J}_a, \eta_a) \approx_s (\mathcal{J}_c, \eta_c)$ via ϕ_1^{ac}, ϕ_2^{ac} and (Θ_{ac}, η_{ac}), as illustrated in Figure 5.1. In what follows, we will first describe the construction of (Θ_{ac}, η_{ac}) and ϕ_1^{ac}, ϕ_2^{ac}. We will then show that the entailment relation $\Theta_{ac} \models \eta_{ac}$ holds by describing the steps to extend any model I_{ac} for Θ_{ac} to a model of η_{ac} by constructing a model I_a of $\mathcal{J}_a$ (or I_c of $\mathcal{J}_c$, respectively)

First, in order to construct Θ_{ac} and η_{ac}, take $\mathcal{J}_b, \eta_b$ and label each position p in their parse trees[4] with variables

- (x, ab) if $\phi_2^{ab}(x) = \mathcal{J}_b|_p$ and $\Theta_{ab}|_p = x$
- (x, bc) if $\phi_1^{bc}(x) = \mathcal{J}_b|_p$ and $\Theta_{bc}|_p = x$

and the analogue for η_b, i.e. we mark nodes in the parse trees of $(\mathcal{J}_b, \eta_b)$ with variables from (Θ_{ab}, η_{ab}) if applicable. Please note that a node in the parse trees may be labelled with two such variable labels $(x, ab), (x', bc)$ if $\phi_2^{ab}(x) = \phi_1^{bc}(x') = \mathcal{J}_b|_p$.

Now (Θ_{ac}, η_{ac}) is obtained from this labelled tree by

- removing all subtrees below nodes labelled with variables $(x, \ldots)$

[4]We use standard notations: $\mathcal{J}|_p$ is the subtree/subexpression of $\mathcal{J}$ at position p, and $\mathcal{J}[s \to t]$ the tree obtained by replacing, in $\mathcal{J}$, the subtree at position s with t.

- turning the variables x in these labels $(x, \ldots)$ into leaf nodes (giving precedence to (x, ab) in the presence of two labels), and
- serialising the result into a justification and entailment template (Θ_{ac}, η_{ac}).

We then construct the mappings ϕ_1^{ac} and ϕ_2^{ac}: if a variable x in (Θ_{ac}, η_{ac}) originates from a labelling (x, ab) in $\mathcal{J}_b$ at position p, then

- $\phi_1^{ac}(x) = \phi_2^{ab}(x)$ and
- $\phi_2^{ac}(x) = \mathcal{J}_b|_p[t_j \to \phi_2^{bc}(y_j)]$ for t_j the positions in $\mathcal{J}_b|_p$ that are labelled with (y_j, bc).

and the analogue for variables originating from labellings (x, bc), and for η_b.

Finally, we need to ensure that the resulting Θ_{ac} indeed entails η_{ac}. We prove that every model I_{ac} for Θ_{ac} is a model for η_{ac} by first showing that, for I_{ac} a model of Θ_{ac}, we can extend I_{ac} to a model I_a for $\mathcal{J}_a$ (or the analogue for $I_c, \mathcal{J}_c$). Given a model $I_{ac} = (\triangle^{I_{ac}}, \cdot^{I_{ac}})$ of Θ_{ac}, we construct I_a such that for every $x \mapsto C$ in ϕ_1^{ac} it holds that $C^{I_a} = x^{I_{ac}}$.

Recall that the expressions in $\mathcal{J}_a$ have to fulfil the side conditions **S1**–**S3** mentioned above in order for this construction to be possible. The construction steps for I_a then are as follows:

- Take a model I_{ac} of Θ_{ac}.
- For each $x \mapsto C$ consider the extension of $x^{I_{ac}} = \{a_1, a_2, \ldots\}$ where $\left|x^{I_{ac}}\right| = k$ (possibly infinite) and take a model I of C with $|C| = 1$ to create k 'copies' of I. This means we obtain a copy I_i of I for each $a_i \in x^{I_{ac}}$.
- For each $a_i \in x^{I_{ac}}$, take I_i (the i-th copy of I) and replace the single element z_i in C^{I_i} with a_i; in particular, replace $(z_i, b) \in r^{I_i}$ with (a_i, b).
- Finally, unite the I_{ac} and the models I_i created in this way by taking their disjoint union [Lut02, p 191], that is, the union of the domains and the union of the respective extensions in the models, to obtain a model I_a for $\mathcal{J}_a$.

With the above steps we have shown that, given any model $I_{ac} \models \Theta_{ac}$, it is possible to construct a model I_a such that $I_a \models \mathcal{J}_a$. Since $\mathcal{J}_a \models \eta_a$, it also holds that $I_a \models \eta_a$. By construction we know that, for every variable x in Θ_{ac} or η_{ac}, it holds that $(\phi_1^{ac}(x))^{I_a} = x^{I_{ac}}$ and thus $I_{ac} \models \eta_{ac}$. This means that every model I_{ac} for Θ_{ac} is a model for η_{ac}, thus the entailment relation $\Theta_{ac} \models \eta_{ac}$ holds. We have therefore shown that we can always construct a template (Θ_{ac}, η_{ac}) for the justifications $\mathcal{J}_a$ and $\mathcal{J}_c$, and thus $\mathcal{J}_a \approx_s \mathcal{J}_c$. □

In order to demonstrate the non-transitivity of subexpression-isomorphism in

the presence of nominals, we consider a simple counter-example:

$$\mathcal{O} = \{C \sqsubseteq \{a\}, C \sqsubseteq \{b\}, \{a\} \sqcap \{b\} \sqcap \leq 1r.C \sqsubseteq D\}$$

which entails $\eta = C \sqsubseteq D$. The only justification for η is the set of all axioms in $\mathcal{O}$. Now modify the ontology to obtain

$$\begin{aligned}\mathcal{O}_a =& \{C \sqsubseteq A, C \sqsubseteq \{b\}, A \sqcap \{b\} \sqcap \leq 1r.C \sqsubseteq D\} \\ \mathcal{O}_b =& \{C \sqsubseteq \{a\}, C \sqsubseteq B, \{a\} \sqcap B \sqcap \leq 1r.C \sqsubseteq D\}.\end{aligned}$$

Both $\mathcal{O}_a$ and $\mathcal{O}_b$ entail η, and we have that $\mathcal{O} \approx_s \mathcal{O}_a$ and $\mathcal{O} \approx_s \mathcal{O}_b$. However, if we substitute the nominals $\{a\}$ and $\{b\}$ with atomic concepts X and Y, respectively, in the template

$$\Theta = \{C \sqsubseteq X, C \sqsubseteq Y, X \sqcap Y \sqcap \leq 1r.C \sqsubseteq D\}$$

we find that the conjunct $\leq 1r.C$ is no longer satisfied by all instances of C (and thus x and y) and Θ does not entail $C \sqsubseteq D$. This shows that the relation $\approx_s$ is not transitive in the presence of nominals.

Finally, note that the substitutions ϕ_1 and ϕ_2 are injective, but not surjective. As we have shown in the proof of 5.2, injectivity of the mappings is a condition for the transitivity of subexpression-isomorphism. However, the mappings are not surjective, as the set of subexpressions in the justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ can be of higher cardinality than the set of class names in Θ (unless the justifications themselves contain no complex expressions). This is illustrated by the following counter-example of non-surjective mappings:

Example 5.4.

$$\begin{aligned}\mathcal{J}_1 &= \{A \sqsubseteq B, B \sqsubseteq C\} \\ \mathcal{J}_2 &= \{A \sqsubseteq B \sqcap \exists r.C, B \sqcap \exists r.C \sqsubseteq D\} \\ \Theta &= \{x_1 \sqsubseteq x_2, x_2 \sqsubseteq x_3\} \\ \phi_1 &= \{x_1 \mapsto A, x_2 \mapsto B, x_3 \mapsto C\} \\ \phi_2 &= \{x_1 \mapsto A, x_2 \mapsto B \sqcap \exists r.C, x_3 \mapsto D\}\end{aligned}$$

The set of *all* subexpressions of $\mathcal{J}_2$ is $\{A, B, \exists r.C, B \sqcap \exists r.C, D\}$. ϕ_2 maps only to a subset of these subexpressions; there exists no mapping for the 'smaller' expressions B and $\exists r.C$. This shows that a substitution ϕ can be non-surjective.

Despite the mappings being non-surjective, we require each subexpression to either have a corresponding variable in Θ which maps to it directly, or to occur as part of a larger subexpression which has a corresponding variable in $\mathcal{J}$. This guarantees that the isomorphic justifications have the same number of subexpressions which correspond to variables in $\mathcal{J}$.

5.2.1 Preferred templates

Given a pair of isomorphic justifications, the template Θ and the substitutions ϕ_1, ϕ_2 are not necessarily unique; there can be multiple possible substitutions, as demonstrated in Example 5.5. The substitutions differ in the level of *granularity*, that is, how large a subexpression a variable in Θ is mapped to. Keeping in mind our eventual goal of helping users cope with justifications, we introduce *preferred templates* which reduce the amount of unnecessary detail in a justification template as far as possible. This means that a preferred template Θ_p contains the smallest possible expressions, i.e. atomic variable names, where possible. In turn, a substitution ϕ_i maps a variable in a preferred Θ_p to the largest possible subexpression in its corresponding justification $\mathcal{J}_i$. Assuming that Θ, Θ_p are valid templates for a pair of justifications $(\mathcal{J}_1, \eta_1)$ $(\mathcal{J}_2, \eta_2)$, and $sig(\Theta) \cup sig(\eta)$ is the set of all variables occurring in a template (Θ, η), we can define preferred templates as follows:

Definition 5.3 (Preferred template)**.** *A template* (Θ_p, η_p) *is a* preferred template *if there is no template* (Θ, η) *such that* $(sig(\Theta) \cup sig(\eta)) \subset (sig(\Theta_p) \cup sig(\eta_p))$.

We illustrate the concept of preferred templates with the following example:

Example 5.5.

$$\mathcal{J}_1 = \{A \sqsubseteq B \sqcap C \sqcap D, B \sqcap C \sqcap D \sqsubseteq E\}$$
$$\mathcal{J}_2 = \{A \sqsubseteq B \sqcap \exists r.C, B \sqcap \exists r.C \sqsubseteq D\}$$
$$\Theta = \{x_1 \sqsubseteq x_2 \sqcap x_3, x_2 \sqcap x_3 \sqsubseteq x_4\}$$
$$\phi_1 = \{x_1 \mapsto A, x_2 \mapsto B, x_3 \mapsto C \sqcap D, x_4 \mapsto E\}$$
$$\phi_2 = \{x_1 \mapsto A, x_2 \mapsto B, x_3 \mapsto \exists r.C, x_4 \mapsto D\}$$

In the above example, the template Θ contains a conjunction $x_2 \sqcap x_3$, which means that only parts of the conjunctions in the justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ are substituted by variables. It is straightforward to see, however, that there exists an alternative Θ_p and substitutions ϕ'_1, ϕ'_2 which substitute larger expressions in the two justifications:

Example 5.6.

$$\Theta_p = \{x_1 \sqsubseteq x_2, x_2 \sqsubseteq x_3\}$$
$$\phi'_1 = \{x_1 \mapsto A, x_2 \mapsto B \sqcap C \sqcap D, x_3 \mapsto E\}$$
$$\phi'_2 = \{x_1 \mapsto A, x_2 \mapsto B \sqcap \exists r.C, x_3 \mapsto D\}$$

The template Θ_p in Example 5.6 has a smaller signature size than Θ; it is therefore the preferred template for this pair of justifications.

5.3 Lemma-isomorphism

While s-isomorphism covers a number of justifications that can be regarded as equivalent due to them requiring the same type of reasoning to reach the entailment, it only applies to justifications which have the same number of axioms. S-isomorphism does not take into account cases where the justifications differ only marginally in some subset, but where the general reasoning may be regarded as similar nonetheless. We therefore introduce the notion of *lemma-isomorphism*, which extends subexpression-isomorphism with the substitution of subsets of justifications through intermediate entailments, so-called *lemmas*. The general motivation behind lemma-isomorphism is demonstrated by the following example:

Example 5.7.

$$\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq C\} \models A \sqsubseteq C$$
$$\mathcal{J}_2 = \{A \sqsubseteq B, B \sqsubseteq C, C \sqsubseteq D\} \models A \sqsubseteq D$$

It is straightforward to see that both $\mathcal{J}_1$ and $\mathcal{J}_2$ require the same type of reasoning from a human user. As the justifications only differ in the length of the atomic subsumption chains that lead to the entailment, we can certainly consider them to be *similar* with respect to *some* similarity measure. However, the two justifications are not considered isomorphic with respect to the definitions for strict isomorphism or subexpression-isomorphism. We therefore introduce a new type of isomorphism which takes into account the fact that subsets of justifications can be replaced with intermediate entailments which follow from them.

A lemma λ for a justification $\mathcal{J}$ is an entailment of a subset S of $\mathcal{J}$. In order to define lemma-isomorphism, we need to introduce two more concepts: *tidy* axiom sets, and justification *lemmatisations*, that is, justifications which are *enriched* with lemmas, as defined by Horridge [Hor11a].

Tidy axiom sets prevent meaningless lemmatisations by ensuring that the set S that generates a lemma is consistent and does not entail synonyms for $\top$ or $\bot$. They are defined as follows [Hor11a, p 252]:

Definition 5.4 (Tidy axiom sets)**.** *A set of axioms S is* tidy *if $S \not\models \top \sqsubseteq \bot$, $S \not\models A \sqsubseteq \bot$ and $S \not\models \top \sqsubseteq A$ for all $A \in sig(S)$.*

The following definition of *summarising lemmatisations* is based on the definition of lemmatisations given by Horridge [Hor11a, p 253], however, with some modifications: first, Horridge's definition considers the *cognitive complexity* of a lemmatisation in order to ensure that a lemmatisation is *easier* to understand than the justification it is based on. As the complexity of a lemmatisation is not relevant for lemma-isomorphism, we drop this condition. Second, the original definition is too weak for use in lemma-isomorphism, as we will show with an example below; thus, we introduce a new condition for the justification subset S which ensures that the lemmas used for the lemmatisation are *summarising*. And finally, we simplify the definition to omit some unneeded details:

Definition 5.5 (Summarising lemmatisation)**.** *Let $(\mathcal{J}, \eta)$ be a justification, let $S = \bigcup S_i$ be a set of* tidy *subsets of $\mathcal{J}$, and let $\Lambda_S = \bigcup \lambda_i$ be a set of lemmas such that $S_i \models \lambda_i$. The set $\mathcal{J}^{\Lambda_S} = (\mathcal{J} \setminus S) \cup \Lambda_S$ is called a* summarising lemmatisation *of $\mathcal{J}$ if*

1. *$\mathcal{J}^{\Lambda_S}$ is a justification for η,*
2. *S_i are pairwise disjoint.*

If clear from the context, a summarising lemmatisation $\mathcal{J}^{\Lambda_S}$ may also be called a *lemmatised justification*.

Proposition 5.3. *Given a $S_i \in S$ that is substituted with a lemma λ_i in a summarising lemmatisation $\mathcal{J}^{\Lambda_S}$, there is no set $S' \subset S_i$ such that $S' \models \lambda_i$.*

Proof. This proposition follows from the minimality condition for justifications: given a justification $(\mathcal{J}, \eta)$, we remove a subset $S_i \subseteq \mathcal{J}$ and replace it with a lemma λ_i such that $(\mathcal{J} \setminus S_i) \cup \{\lambda_i\}$ is a justification for η. If there exists a subset $S' \subset S_i$ such that $S' \models \lambda_i$, then it also holds that $(\mathcal{J} \setminus S_i) \cup S' \models \eta$. Therefore, the axioms $S_i \setminus S'$ are redundant with respect to the entailment η, which violates the minimality condition for $\mathcal{J}^{\Lambda_S}$. □

Given the definitions for lemmatisations, we can now define lemma-isomorphism as an extension to subexpression-isomorphism:

Definition 5.6 (Lemma-isomorphism)**.** *Two justifications $(\mathcal{J}_1, \eta_1)$, $(\mathcal{J}_2, \eta_2)$ are* ℓ-isomorphic *($(\mathcal{J}_1, \eta) \approx_\ell (\mathcal{J}_2, \eta)$) if there exist* summarising *lemmatisations $\mathcal{J}_1^{\Lambda_{S1}}$, $\mathcal{J}_2^{\Lambda_{S2}}$ which are s-isomorphic: $\mathcal{J}_1^{\Lambda_{S1}} \approx_s \mathcal{J}_2^{\Lambda_{S2}}$.*

Lemma-isomorphism as defined above carries two undesirable properties: first, the lemmatised justifications might differ strongly from the original justifications; in the most extreme case, the lemmatisation of a justification can be the entailment itself. We therefore have to introduce a constraint that ensures that lemmatisation must be understandable to a user; thus, we only permit lemmatisations which are based on *obvious* steps, as described in the next section.

Second, we cannot guarantee that lemma-isomorphism using arbitrary lemmas is in fact transitive. Hence, together with the *obviousness* of the lemmas, there are several design decisions to be made when choosing legitimate lemmatisations:

1. Allow arbitrary non-summarising lemmas.
2. Allow only summarising lemmas. While we cannot show that this preserves transitivity, substituting complete subsets with their lemmas seems more adequate for making lemmatisations easily understandable.
3. Allow only specific types of lemmas, e.g. entailments of atomic subsumption chains. We take this approach below in order to restrict lemmatisations to obvious proof steps.
4. Allow only specific types of atomic subsumption chains, e.g. *maximal* atomic subsumption chains. As we will show in Section 5.3.1, this is necessary for a practical implementation of lemma-isomorphism.

We will first discuss the issue of obvious steps for lemmatisations, before selecting a suitable type of lemmas—entailments of atomic subsumption chains—which are both cognitively adequate and can be computed in practical time.

5.3.1 Lemmatisations and obvious steps

The notion of *obvious logical inferences* [Dav81] describes how proof steps which are *obvious* for a reader can be omitted without losing information that may be important for understanding the proof. We adapt this notion of obvious steps in order to allow only lemmatisations which are easily understandable for a human reader, that is, lemmas which follow from a subset of a justification in a straightforward fashion.

While research on human understanding of description logic is fairly limited, the works by Horridge [HBPS11a] and Nguyen [NPPW12a] provide us with two different models for determining the complexity of justifications and justification subsets, respectively, and thus the obviousness of lemmas.

While the rankings determined by Nguyen et al. [NPPW12a] seem to cover the most frequently occurring justification subset patterns, they do not include atomic subsumption chains, which have been found to be one of the most prevalent and easiest recognizable patterns [HBPS11a]. We therefore focus on atomic subsumption chains for the purpose of demonstrating the effects of lemma-isomorphism.

Atomic subsumption chains

Given an atomic subsumption chain of the type $S = \{A_0 \sqsubseteq A_1, A_1 \sqsubseteq A_2 \ldots A_{n-1} \sqsubseteq A_n\}$ in a justification, often only the relation between the subclass A_0 in the first axiom and the superclass A_n in the last axiom is relevant for understanding the how the chain contributes to the justification. That is, the subsumption chain produces a summarising lemma $A_0 \sqsubseteq A_n$ which is the only piece of information required for the justification. The step from the subclass to the final superclass was found to be *obvious* for OWL users who were presented with justifications containing ordered and indented subsumption chains of this type [HBPS11a]. Therefore, a suitable lemmatisation of a justification containing an atomic subsumption chain is the justification obtained by substituting the chain with its conclusion in the form of a single axiom $A_0 \sqsubseteq A_n$.

Maximal atomic subsumption chains Recall that we require the lemmatisations of atomic subsumption chains in a justification to be summarising, that is, the subsumption chain can be substituted by a single axiom and the entailment of the justification still holds. However, axioms that appear to be a single atomic subsumption chains may not always be such a *maximal* chain: if a chain $A_0 \sqsubseteq \ldots \sqsubseteq A_n$ can be substituted by a summarising lemma λ, and there exists a longer chain $A_0 \sqsubseteq \ldots \sqsubseteq A_{n+1}$, then the longer chain cannot always be substituted by a summarising lemma λ'. It is certainly possible for a set of axioms to form an apparently maximal subsumption chain in which only a subset of the chain can be substituted by a summarising lemma, as shown in the following example:

Example 5.8.

$$\mathcal{J} = \{A \sqsubseteq \exists r.C, A \sqsubseteq B, B \sqsubseteq C, C \sqsubseteq D, C \sqcap \exists r.D \sqsubseteq E\} \models A \sqsubseteq E$$

Take the justification $\mathcal{J}$ in Example 5.8. The axiom set $\{A \sqsubseteq B, B \sqsubseteq C, C \sqsubseteq D\}$ is a maximal atomic subsumption chain in $\mathcal{J}$ which can be substituted by its entailment $A \sqsubseteq D$. It is clear to see, however, that the entailment does not hold in this lemmatisation as the important lemmas $A \sqsubseteq C$ and $C \sqsubseteq D$ would be lost. We can only substitute part of the subsumption chain, namely $\{A \sqsubseteq B, B \sqsubseteq C\}$ with its entailment $A \sqsubseteq C$; we call the axiom set $\{A \sqsubseteq B, B \sqsubseteq C\}$ a non-maximal chain.

Non-maximal subsumption chains are problematic as they can to lead to non-transitivity of lemma-isomorphism, as is shown in the following example:

Example 5.9.

$\mathcal{J}_a \models X \sqsubseteq Y$	$\mathcal{J}_b \models X \sqsubseteq Y$	$\mathcal{J}_c \models X \sqsubseteq Y$
$X \sqsubseteq (A_0 \sqcap A_2) \sqcap (\forall r.A_0 \sqcap \forall s.A_4)$	$X \sqsubseteq (B_0 \sqcap B_2) \sqcap (\forall r.B_0 \sqcap \forall s.B_4)$	$X \sqsubseteq (C_0 \sqcap C_2) \sqcap (\forall r.C_0 \sqcap \forall s.C_4)$
$(A_1 \sqcap A_5) \sqcap (\forall r.A_3 \sqcap \forall s.A_5) \sqsubseteq Y$	$(B_1 \sqcap B_5) \sqcap (\forall r.B_3 \sqcap \forall s.B_5) \sqsubseteq Y$	$(C_1 \sqcap C_5) \sqcap (\forall r.C_3 \sqcap \forall s.C_5) \sqsubseteq Y$
$A_0 \sqsubseteq A_1$	$B_0 \sqsubseteq B_1$	$C_0 \sqsubseteq C_3$
$A_2 \sqsubseteq A_5$	$B_1 \sqsubseteq B_2$	$C_4 \sqsubseteq C_5$
	$B_2 \sqsubseteq B_3$	
	$B_3 \sqsubseteq B_4$	
	$B_4 \sqsubseteq B_5$	

In the above example, the three axiom sets are justifications for $X \sqsubseteq Y$. It holds that $\mathcal{J}_a \approx_\ell \mathcal{J}_b$ and $\mathcal{J}_b \approx_\ell \mathcal{J}_c$ via lemmatisations $\mathcal{J}_b^{\lambda_1}$ and $\mathcal{J}_b^{\lambda_2}$, respectively. While the set of atomic subsumption axioms in $\mathcal{J}_b$ appears to be a single maximal

chain, it cannot be substituted with a single lemma without breaking the entailment. In $\mathcal{J}_b^{\lambda_1}$ the atomic subsumption chain $B_0 \dots B_5$ is substituted with the two lemmas $\lambda_{11} = B_0 \sqsubseteq B_1, \lambda_{12} = B_2 \sqsubseteq B_5$, which makes the lemmatisation strictly isomorphic to $\mathcal{J}_a$. The analogue holds for $\mathcal{J}_b^{\lambda_2}$ which is strictly isomorphic to $\mathcal{J}_b$ via the two lemmas $\lambda_{21} = B_0 \sqsubseteq B_3, \lambda_{22} = B_4 \sqsubseteq B_5$. However, $\mathcal{J}_a$ and $\mathcal{J}_c$ are clearly not s-isomorphic. This shows that lemma-isomorphism which allows the substitution of non-maximal chains with lemmas can lead to non-transitivity of the relation.

While we have not yet proved that maximal-chain lemma-isomorphism is guaranteed to be transitive, restricting the lemmatisations to maximal chains removes one possible cause of non-transitivity. However, the transitivity of lemma-isomorphism with maximal atomic subsumption chains remains an open question; for the time being, we treat all results of the lemma-isomorphism implementation as possible approximations. This means that it is possible that some sets of justifications may be considered to be non-isomorphic when they actually are. Hence, we may potentially over-estimate the logical diversity of a corpus of justifications and under-estimate the effects of lemma-isomorphism, which we consider acceptable for the purpose of demonstrating the concept of lemma-isomorphism.

In summary, for the purpose of demonstrating lemma-isomorphism in this thesis, we restrict the admissable lemmatisations of a justification $(\mathcal{J}, \eta)$ to

- atomic subsumption chains of type $A_0 \sqsubseteq \dots \sqsubseteq A_n$
- that are summarising, i.e. $\mathcal{J} \setminus S \cup \{\lambda\} \models \eta$ for S a set of axioms of type $A_0 \sqsubseteq \dots \sqsubseteq A_n$ and $\lambda = A_0 \sqsubseteq A_n$
- that are maximal, i.e. given a chain $A_0 \sqsubseteq \dots \sqsubseteq A_n$, there exists a summarising $\lambda = A_0 \sqsubseteq A_n$ such that $\mathcal{J} \setminus S \cup \{\lambda\} \models \eta$.

5.4 Equivalence and superfluity

For the purpose of analysing the diversity of reasons why an entailment holds in an ontology, we need to take into account another aspect of justifications, namely the superfluousness of subexpressions in an axiom. Since justifications are sets of axioms *as they are asserted in the ontology*, it is possible for such an axiom to contain parts which are not relevant with respect to the entailment. This can have several possible effects: 1) A justification can contain *multiple* reasons for an entailment to hold. 2) Two justifications can have exactly the same reason why

an entailment holds and only differ in their superfluous parts; this results in *fewer* logical reasons than material justifications. 3) A justification can have parts which interact with *other* axioms in the ontology to form entirely *new* justifications. These phenomena are also known as facets of *justification masking* [HPS10a] which was introduced in Chapter 2. In order to take into account the effects of masking in justifications and to eliminate potentially distracting superfluous expressions, Horridge et al. [HPS08] defined *laconic* justifications which we also discussed in Chapter 2.

It is clear to see that superfluity can lead to justifications being considered non-isomorphic, despite them being identical 'in principle' and requiring the same reasoning strategy. As a simple example, take the following two justifications:

$$\mathcal{J}_1 = \{A \sqsubseteq B \sqcap C, B \sqsubseteq D\} \models A \sqsubseteq D$$
$$\mathcal{J}_2 = \{A \sqsubseteq B, B \sqsubseteq D\} \models A \sqsubseteq D$$

Both justifications only require atomic subsumption chain reasoning; the conjunct C in $\mathcal{J}_1$ is superfluous as it does not affect the entailment. However, the justifications could not be considered to be isomorphic with respect to any of our relations. Note that the superfluity itself is not the culprit here, since a similarly superfluous conjunct in J_2 would cause the justifications to be (strictly or subexpression-) isomorphic nonetheless. It is the superfluity (or lack thereof) in different positions which causes the structural similarity of the justifications to be concealed.

We know that non-laconic justifications are highly prevalent in OWL ontologies, as shown by a study of 72 BioPortal ontologies in which over 82% of the justifications were found to contain superfluous expressions [Hor11a]. This indicates that in order to obtain a more accurate picture of the logical diversity of a set of justifications we may choose to identify isomorphic justifications based on their laconic versions. Note that due to internal masking, a justification can have *multiple* laconic versions. This means that a necessary condition for the justifications to be considered isomorphic is that *all* their laconic versions have to be isomorphic.

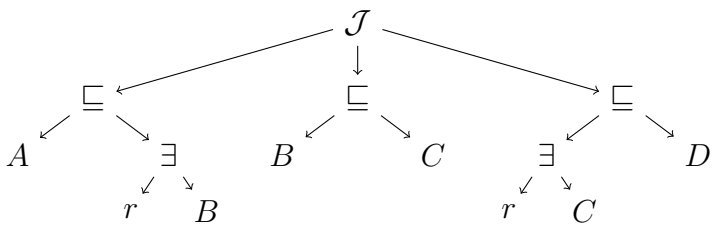


Figure 5.2: Parse tree of $\mathcal{J} = \{A \sqsubseteq \exists r.B, B \sqsubseteq C, \exists r.C \sqsubseteq D\} \models A \sqsubseteq D$.

5.5 Implementing an isomorphism checker

Having introduced the basic notions of isomorphism in the previous section, we will now discuss our approach to implementing an isomorphism checker which covers all three types of isomorphism. We outline the algorithm for detecting isomorphism which is based on a comparison between the parse trees of the justifications to be matched, and discuss some of the basic optimisations that can be applied to improve the performance of the isomorphism checker. Finally, we describe how the j-graph introduced in Chapter 4 can be extended to include justification templates for structurally isomorphic justifications.

5.5.1 Algorithm and implementation

The general idea behind the implementation of a program to check whether two justifications are isomorphic is the comparison of the *parse trees* of the two justifications. Each axiom in a justification is parsed into its parse tree where the nodes are labelled with either entity names (properties or classes), OWL constructors, or integers (in the case of cardinality restrictions). The parse tree of a justification simply contains each axiom parse tree as child of a blank root node. To simplify the algorithm, the respective entailments are treated *separately* from the justification parse trees, as this allows us to treat every axiom subtree in the justification parse tree equally. An example of a justification parse tree (without the respective parse tree of the entailment) is shown in Figure 5.2.

Given the parse trees of two justifications $(\mathcal{J}_1, \eta_1)$, $(\mathcal{J}_2, \eta_2)$, the algorithm answers 'true' if the justifications are isomorphic wrt. one of the given definitions for isomorphism, and 'false' otherwise. While there exist algorithms for tree-isomorphism (e.g. [ST99]), we chose to implement a naive matching algorithm,

as this would allow us to extend the algorithm from strict to subexpression-isomorphism in a straightforward fashion.

For strict isomorphism, the algorithm answers 'true' if it can construct a mapping ϕ (that is, a set of pairs $(n1, n2)$) which maps nodes $n1$ in the parse trees of $\mathcal{J}_1$ and η_1 to nodes $n2$ in $\mathcal{J}_2$ and η_2 such that $\phi(\mathcal{J}_1) = \mathcal{J}_2$ and $\phi(\eta_1) = \eta_2$. For subexpression-isomorphism, we deviate from the given definition and attempt to find only a single mapping ϕ which maps subtrees in $\mathcal{J}_1$ and η_1 to subtrees in $\mathcal{J}_2$ and η_2, respectively. Finally, as lemma-isomorphism is based on subexpression-isomorphism, we first lemmatise the justifications, then apply the algorithm for detecting subexpression-isomorphism to the lemmatised justifications.

An outline of the algorithm for strict isomorphism and its subroutines are given in Algorithms 5.1 and 5.2, whereby childCount(), removeChild(), and getChildAt() can be interpreted as standard operations on a tree structure. The algorithm traverses the parse trees of $\mathcal{J}_1$ and $\mathcal{J}_2$ (denoted by $e1$ and $e2$ in the pseudocode) depth-first, left to right, and attempts to construct a set *Mappings* of candidate mappings. It checks at each node whether the nodes at this position in the parse trees are compatible in the context of the current mapping ϕ. Note that the parse trees of the entailments are not considered here; they only come into play in the final call to isCorrectMapping()

The subroutine compatible() returns 'true' if a) the nodes are both labelled with the same constructor (sameOperator() holds true) and have the same number of children, b) the nodes are leaf nodes and respect the existing mapping ϕ, that is, the exact pair $t1, t2$ either exists already in ϕ or neither $t1$ nor $t2$ occur in ϕ. Otherwise, compatible() returns 'false'. If a match is found (i.e. the nodes are compatible), then we add the pair to the existing mapping and continue with the children and the siblings of $t1$. Finally, if a set of candidate mappings is found, the substitutions are applied to the justifications and entailments and an entailment check is performed in order to verify whether the substitutions preserve the entailment. If such a mapping is found, the algorithm returns 'true'.

5.5.2 Optimisations

There are several optimisations to the implementation of the generic tree comparison algorithm which can be used to reduce the number of comparisons, thus improving performance. The optimisations can be categorised into a) filtering prior to starting the tree comparison, and b) early termination of the comparison

Algorithm 5.1 EQUIVALENT$(e1, e2)$

```
1: t1 ← getParsetree(e1)
2: t2 ← getParsetree(e2)
3: φ ← ∅
4: Mappings ← match(t1, t2, φ)
5: for φ ∈ Mappings do
6:     if isCorrectMapping(φ, t1, t2) then
7:         return true
8:     end if
9: end for
10: return false
```

algorithm. Some of these optimisations depend on the respective isomorphism type that is being checked for.

Strict isomorphism optimisations Before starting the tree comparison, we can filter out non-matching justifications based on

1. the number of axioms in the justifications
2. the number of different axiom types
3. the signature size of each justification
4. the number of different constructor types used in the axioms.

We only perform a full tree comparison if these values match for both justifications.

Subexpression- and lemma-isomorphism optimisations As subexpression-isomorphism allows mappings between complex expressions, the types and numbers of constructors can differ between the two justifications. Thus, we can only apply optimisations 1) and 2) which compare the number and types of axioms. As lemma-isomorphism reduces to subexpression-isomorphism after the lemmatisation stage, we can apply the same optimisations.

General optimisations In addition to the specific optimisations described above, the most obvious cheap check is a check for equality of the justifications before calling the tree algorithm. Further, we can apply an ordering on the axioms in the (lemmatised) justification when generating the parse tree. As only axioms of the same type can potentially be considered isomorphic, ordering the children in the justification tree based on their axiom type prevents the algorithm

Algorithm 5.2 Subroutines to Algorithm 5.1

```
1: function Match(t1, t2, φ)
2:     if (t1, t2) ∈ φ then
3:         return matchChildren(t1, t2, φ)
4:     end if
5:     if compatible(t1, t2, φ) then
6:         φ ← φ ∪ {(t1, t2)}
7:         return matchChildren(t1, t2, φ)
8:     end if
9:     return ∅
10: end function

11: function MatchChildren(t1, t2, φ)
12:     if childCount(t1) = 0 then
13:         return φ
14:     end if
15:     Mappings ← ∅
16:     FirstChild ← getChildAt(t1, 0)
17:     ReducedT1 ← removeChild(t1, FirstChild)
18:     for Child ∈ t2 do
19:         ReducedT2 ← removeChild(t2, Child)
20:         for φ ∈ match(t1, t2) do
21:             Mappings ∪ matchChildren(ReducedT1, ReducedT2, φ)
22:         end for
23:     end for
24:     return Mappings
25: end function

26: function Compatible(t1, t2, φ)
27:     if childCount(t1) ≠ childCount(t2) then
28:         return false
29:     else if sameOperator(t1, t2) then
30:         return true
31:     else if respectsMapping(t1, t2, φ) then
32:         return true
33:     end if
34:     return false
35: end function
```

from matching incompatible axiom types in the first place, thus gaining a small advantage over a random comparison between axiom parse trees.

5.5.3 Limitations due to syntactical differences

Many constructors and axiom types in OWL have an abstract, frame-like syntax, which can be used interchangeably with their symbolic description logic syntax. For example, one of the most common variants can be found for domain axioms, since $domain(r, C)$ can also be written as $\exists r.\top \sqsubseteq C$.

Beyond notational variants, we also need to take into account the equivalence of expressions. For example, the expression $\neg\exists r.A$ is equivalent to $\forall r.(\neg A)$, and we can certainly imagine ontology developers unknowingly using both notations in an ontology.

Given our current equivalence relations, justifications containing equivalent axioms with alternative syntaxes or equivalent expressions would not be considered isomorphic. The question arises whether we can (or should) consider those alternative notations to be isomorphic.

On the one hand, the semantics of the alternative notations are identical; for the purpose of analysing the logical diversity of a corpus of ontologies, the material form of an axiom is clearly not relevant. By introducing a *normalisation* step prior to computing the equivalence classes on a corpus of justifications, we can convert each such axiom into our preferred notation and partially eradicate the issue of seemingly diverse justifications.

On the other hand, however, we may argue that, from a user perspective, there exists a clear difference between the syntactical variants. In description logic syntax, a justification of type $\{A \sqsubseteq \exists r.B, \exists r.\top \sqsubseteq C\} \models A \sqsubseteq C$ can be solved by a human user through simple pattern matching, as the RHS of the first axiom closely resembles the LHS of the second axiom. While it does require the user to understand that $\exists r.B$ is subsumed by $\exists r.\top$, the similarity between the expressions can support understanding the relation between the axioms. Comparing this to the more abstract notation $\{A \sqsubseteq \exists r.B, domain(r, C)\} \models A \sqsubseteq C$, the user first needs to interpret the domain axiom correctly, which in turn seems to have little connection to the first axiom. This can potentially complicate the process of understanding the justification, making the abstract syntax quite different from the symbolic notation. A counter-argument for this is the fact that pointing out the equivalence of two different looking expressions may *help* the

user in understanding unfamiliar notation; this, however, is more of a general representational issue.

Thirdly, taking into account the experience or preferences of a user, a normalisation towards the user's preferred notation may in fact be beneficial for supporting their understanding of justifications. Furthermore, in OWL editors such as Protégé 4, users generally only encounter the abstract syntax, with no obvious facilities to construct subsumption axioms with a complex LHS. This supports the argument *for* a normalisation of syntactical variants to match the preferred syntax of a user.

We conclude that, while syntactical variants may differ in their complexity for human users, the normalisation of such equivalent notations can give us a more homogeneous picture of the logical diversity of a corpus of justifications.

5.5.4 Extending the j-graph

Extending the j-graph we have introduced in Chapter 4 to take into account isomorphism relations between sets of justifications can be done in several different ways: first, we could substitute the isomorphic justifications with an entirely new node labelled with the respective template, preserving the incoming and outgoing edges of the justification nodes. Second, we can label each justification node with its template. And third, we can create a new template node for each template with outgoing edges to the corresponding isomorphic justifications.

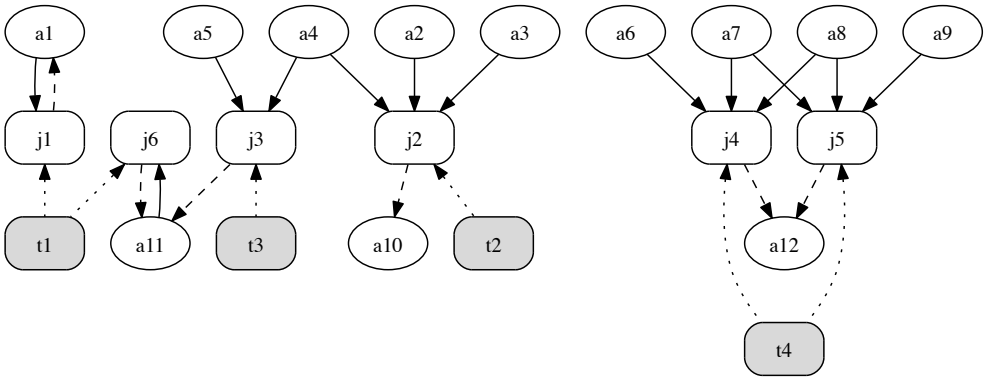


Figure 5.3: An extended j-graph containing four template nodes $t1$ through $t4$.

Since it it desirable to preserve the relationship between the justification axioms, justifications, and their entailments, a complete substitution as suggested by the first approach seems unsuitable. Adding an additional template label to each justification node may be a straightforward solution, but it does not give us direct access to metrics such as the number of templates in the j-graph, or the number of justifications for each template. We therefore choose to represent isomorphic justifications by a newly introduced template node; this allows us to conveniently count the number of templates which corresponds to the number of nodes labelled with a template, and identify the number of justifications for each template, which is simply the out-degree of a template node. In addition, this approach lets us represent template nodes for each isomorphism type in the graph, without running risk of adding too many labels to a single node.

Figure 5.3 shows the example j-graph from Figure 4.2 extended with an additional four template nodes for one type of isomorphism. From left to right, the grey shaded nodes represent templates for justifications $j1$ and $j6$ ($t1$), $j2$ ($t2$), $j3$ ($t3$), and $j4$ and $j5$ ($t4$), respectively. The one piece of information we are missing out is the relation between isomorphic *axioms* in a set of justifications. It would certainly be possible to further extend the graph with additional axiom template nodes for each justification template, with outgoing edges to the justification template node; however, in order to avoid cluttering of the graph, we omit this information for now.

5.6 Summary and conclusions

Motivated by the apparent structural similarity of distinct justifications, we introduced two new equivalence relations that extend strict justification isomorphism: subexpression-isomorphism covers justifications which contain complex expressions that could be substituted by variable names, that is, the semantics of the expression is not relevant to the entailment. Lemma-isomorphism goes one step further and allows us to consider justifications to be isomorphic if there exist *lemmatisations* of the justifications which are subexpression-isomorphic. In this chapter, we presented definitions for these new relations and presented a proof for the (non-obvious) transitivity of the subexpression-isomorphism in description logics without nominals. We gave an overview of an algorithm to detect isomorphism between two justifications and discussed possible optimisations for

an implementation.

The work presented in this chapter is an important step on the way to gaining a better understanding of how we can define and detect similarity between justifications in OWL ontologies. This is relevant in two respects: first, the types and numbers of justification templates for a finite entailment set of an ontology provides us with a metric to measure the logical diversity—and the logical uniformity—of an ontology, which adds another aspect to our suite of metrics of the justificatory structure of OWL ontologies. And second, if confronted with multiple justifications, these relations allow us to point out structural similarities to the user, thus potentially reducing the effort required to understand all justifications; further, subexpression- and lemma-isomorphism reduces the general diversity of expressions and axioms in a justification, which has a similar effect as 'laconicising' a justification in order to remove superfluous parts. A detailed discussion of how we can exploit isomorphism in the debugging process follows in the next chapter.

Chapter 6

Justification comprehension

In the previous chapters we introduced the basic elements of the justificatory structure of an OWL ontology, such as varying degrees of overlap between justifications, and justification isomorphism. In this chapter, we will characterise different scenarios of how OWL ontology users encounter justifications, and how justificatory structure plays a role in those encounters. There are several possible scenarios in which ontology users could use justifications:

- Information: understanding why an entailment holds in an ontology, e.g. for learning purposes or to verify the correctness of the ontology.
- Debugging: modifying an ontology such that one or several erroneous entailments are no longer entailed.
- Analysis: obtaining metrics about the justificatory structure of an ontology, e.g. for comparing two ontologies.
- Ontology comprehension: understanding the structure of an ontology, e.g. for learning purposes or to be able to integrate or modify the ontology.

The term *ontology comprehension* is frequently used in a rather broad sense to refer to the user understanding entities and their (intended) relations in an ontology in order to be able to use them correctly [GWS07, BSP09]; however, ontology comprehension is neither a clearly defined task, nor is *understanding* easy to measure; see, for example, [HBPS11a] for a discussion of the challenges faced when designing user studies on justification understanding. In contrast, generating justification-based metrics does have a clearly defined outcome (the correct metrics), but does not require any additional user support beyond the suitable representation of those metrics. Using justifications for information purposes is very similar to debugging; however, rather than *all* justifications, it often only requires small numbers of (or even just one) justifications to provide the required information to the user. Thus, we focus on the case of *debugging* erroneous entailments which is a clearly defined task with a *measurable* goal.

In the context of ontology debugging we now ask how the justificatory structure of an ontology can be exploited to improve the debugging and repair process for OWL ontology developers. While we commonly speak of *the debugging process*, it is clear to see that this is not a single task, but rather that there are many different facets to the act of detecting errors in an ontology and modifying it to remove these errors. Thus, the measures taken to improve debugging support depend on the task, e.g. whether the user wants to repair *one* or *multiple* entailments, the occurrence of *single* or *multiple* justifications, the repair strategy (*rewriting* or *removing* axioms) as well as the technical background and goal of the user, such as *understanding* the sources of error versus finding a 'quick fix'.

The central focus point of this thesis has been the occurrence of multiple justifications in OWL ontologies, and how we can 'reduce user effort' in a situation where a user encounters multiple justifications. In this chapter, we will define the notion of a *successful* debugging solution which then allows us to pin down what exactly we mean by the *effort* a user has to apply in the debugging process, and how we can measure this effort.

Given these user scenarios, we will then discuss approaches to improving debugging support using the aspects of justificatory structure we have introduced in the previous chapters. The goal of this chapter is to gain a better understanding of the various situations in which users encounter justifications, and to provide an analysis of the impact potential interventions have on a user's effort when dealing with justifications.

6.1 Debugging problems

In this section, we discuss the notion of a *debugging problem* and the various scenarios in which users encounter justification in the debugging process. These scenarios include both the number of entailments and justifications in the given task, as well as the relations between the justifications, if applicable. We will then define the *success* of a debugging problem based on a *minimum loss solution* measure for debugging a set of entailments.

6.1.1 Defining debugging problems

Our definition of a debugging problem is based on the commonly used informal notion of ontology debugging: given an ontology and some erroneous entailments,

rewrite or remove axioms in the ontology such that the errors are removed. First, recall that in this context we focus on *finite* entailment sets, that is, we only consider a finite entailment set $\varepsilon_{\mathcal{O}}$ of an ontology $\mathcal{O}$ of a specific type, such as the set of entailed direct atomic subsumptions involving satisfiable and unsatisfiable classes. As we discussed in Chapter 3, we can partition $\varepsilon_{\mathcal{O}}$ into the set of *unwanted* entailments $\varepsilon^-_{\mathcal{O}}$ and the remainder, the set of *wanted* entailments $\varepsilon^+_{\mathcal{O}}$.

In order to allow us to talk about the effort required to solve a debugging problem, we separate the informal notion of debugging into its two aspects: the debugging *input* and the debugging *task*. Generally, the debugging input consists of an ontology $\mathcal{O}$ and a set $\varepsilon^-_{\mathcal{O}}$ of unwanted entailments which the user considers to be erroneous. The debugging task is to find a *solution* to the debugging problem. A solution is simply a modification *mod* such that applying *mod* to $\mathcal{O}$ yields a new ontology $mod(\mathcal{O}) \subseteq \mathcal{O}$ which entails none of the axioms in $\varepsilon^-_{\mathcal{O}}$. It is clear to see, however, that accepting *any* such solution can lead to undesirable effects: in the most extreme case, simply removing *all* axioms from the ontology would be a solution for a debugging problem.

We therefore place an additional constraint on the acceptable solutions to a debugging problem: a modification *mod* must not only remove all errors, but also try to *preserve* the correct entailments in $\varepsilon^+_{\mathcal{O}}$; that is, an acceptable solution is a *minimum loss* solution. Since it may not be possible to preserve *all* entailments in $\varepsilon^+_{\mathcal{O}}$ (since, for example, some wanted entailments might entail unwanted ones), we loosen this restriction to preserve a *maximal subset* ε^* of $\varepsilon^+_{\mathcal{O}}$. A debugging problem then considers not only the ontology and the set of unwanted entailments, but also includes the set of correct entailments $\varepsilon^+_{\mathcal{O}}$, and the debugging task is extended to finding a minimum loss solution.

In summary, a debugging input $D = \langle \mathcal{O}, \varepsilon^+_{\mathcal{O}}, \varepsilon^-_{\mathcal{O}} \rangle$ consists of an ontology $\mathcal{O}$ and two sets of entailments $\varepsilon^+_{\mathcal{O}}$ and $\varepsilon^-_{\mathcal{O}}$, such that $\mathcal{O} \models \alpha$ for all $\alpha \in \varepsilon^+_{\mathcal{O}} \cup \varepsilon^-_{\mathcal{O}}$. The debugging task is to find a minimum loss solution to the given debugging input:

Definition 6.1. *A* solution *to a debugging problem D is a modification mod such that $mod(\mathcal{O}) \subseteq \mathcal{O}$ and $mod(\mathcal{O}) \not\models \alpha$ for all $\alpha \in \varepsilon^-_{\mathcal{O}}$. A* mimimum loss solution *to a debugging problem D is a modification mod such that*

- *mod is a solution to D,*
- *$mod(\mathcal{O}) \models \alpha$ for all $\alpha \in \varepsilon^*$ where $\varepsilon^* \subseteq \varepsilon^+_{\mathcal{O}}$,*
- *and there is no solution mod' such that $mod'(\mathcal{O}) \models \alpha'$ for all $\alpha' \in \varepsilon'_{\mathcal{O}}$ where $\varepsilon'_{\mathcal{O}} \subset \varepsilon^*$.*

Note that *in general* we may consider a modification *mod* to be a series of manipulation steps including axiom or subexpression rewritings, removals, or additions. However, for the purpose of defining debugging effort in this chapter, we restrict modifications to the *removal* of axioms. This is in line with the concept of finding minimal repairs (i.e. finding a minimal hitting set across a set of justifications and removing it) and allows us to fix the subset relation between the original ontology $\mathcal{O}$ and the repaired ontology $\mathcal{O}'$ which would not be possible if we allowed additions and rewritings. Fixing this subset relation is necessary to allow us to preserve not only the entailments in $\varepsilon^+_{\mathcal{O}}$, but also any other entailments in the deductive closure of $\mathcal{O}$ without explicitly specifying them in $\varepsilon^+_{\mathcal{O}}$. Otherwise, rewriting $\mathcal{O}$ to yield $\varepsilon^+_{\mathcal{O}}$ would always be the considered the ideal solution, but this could lead to an ontology that is vastly different from the original $\mathcal{O}$, which is obviously undesirable.

Finally, we can pin down the notion of a successful solution of a debugging problem: if the user can find a minimum loss modification *mod*, we consider the debugging problem to be solved *successfully*. A debugging problem is *not solved* if after applying a modification it still holds that $\mathcal{O} \models \varepsilon^-_{\mathcal{O}}$ for some entailments in $\varepsilon^-_{\mathcal{O}}$, and solved *unsucessfully* if the applied modification is not a minimum loss modification, that is, more entailments were removed from $\varepsilon^+_{\mathcal{O}}$ than necessary. Of course, there are other aspects to a successful and *good* repair strategy, such as the cognitive effort and time required to find a specific solution, which we have not yet considered in this definition.

6.1.2 Justification encounters

Now that we have defined what we mean by a *debugging problem*, we proceed with an analysis of the different scenarios in which single and multiple justifications can occur in debugging problems. This includes a description of (simple) reading strategies users apply in order to find a suitable modification for a given scenario.

1. **Single entailment**
   (a) **Single justification** There exists only a single justification for the entailment.
   Strategy The entailment is repaired by removing one erroneous axiom (or several, if the error is caused by a set of incorrect statements) in the justification.
   (b) **Multiple justifications** There exist j distinct justifications for the

entailment, some of which may have common axioms.

Strategy Inspect and remove an axiom from every justification individually in a linear fashion. The user might attempt to find common axioms in order to obtain a smaller repair, for example by jumping back and forth between the justifications and memorising which axioms are shared.

2. **Multiple entailments**
   (a) **Single justification** Each of the k entailments has exactly one justification.

   Strategy Consider each entailment and its justification in isolation. If the tool re-computes the entailments after an axiom removal, a single modification *might* affect other justifications, thus reducing the total number of justifications to inspect. Otherwise, the user has to inspect every justification once and remove each incorrect axiom.

   (b) **Multiple justifications** Each of the k entailments has j_i justifications.

   Strategy Inspect and modify every justification for each entailment individually. Again, the user might attempt to find common axioms between the justifications for a single entailment by jumping (scrolling) back and forth between the justifications. Current explanation tools (e.g. in Protégé 4), however, do not support switching between the justifications for different entailments.

In summary, the number of justifications a user encounters using a straightforward linear reading strategy is determined as follows: given a set of k entailments where the i-th entailment has j_i justifications, the number of justifications requiring inspection is $\sum_{1 \leq i \leq k} j_i$.[1]

6.2 Measuring effort

Our main goal is the reduction of *user effort* in the debugging process when confronted with justifications for single or multiple entailments. While we have some intuitions about the notion of user effort—the time taken to find a repair,

[1] For legibility reasons, we will drop the limit $1 \leq i \leq k$ from the sum in the remainder of this chapter and generally assume that i ranges over the number of entailments.

the cognitive difficulties in finding a repair—we have not yet pinned down what exactly we mean when we talk about the effort required to solve a debugging problem, and how we can measure such effort. In what follows, we outline the different facets of user effort, and define a simple model for measuring effort for a given debugging problem.

6.2.1 The complexity of individual justifications

The *cognitive complexity* of OWL justifications has gained some attention in recent years, with some investigations into typical errors users make [RCVB09, CRVBP09] and user studies to test directly which types of justifications users find easy or difficult to deal with [HPS09, HBPS11b, NPPW12a]. For example, the study presented by Nguyen et al. [NPPW12a] ranks frequently occurring axiom subsets in justifications according to the number of study participants who correctly interpreted the natural language representation of the justification. However, the authors do not further investigate what properties cause these justifications to be difficult to understand for the study subjects.

In contrast, the complexity model constructed by Horridge et al. [HPS09] is based on a number of justification features and can be applied to any OWL justification, resulting in a complexity score which ranges from 0 ('very easy') to 2000 ('very hard') and beyond for 'naturally occurring' justifications. The model components are based on features such as the different axiom types and class constructors found in the justification and certain *phenomena*, for example whether the justification entails $\top \equiv A$ for some class A in its signature.

Representation of justifications We have to bear in mind, however, that the difficulty of understanding a justification does not only depend on its *intrinsic* complexity (i.e. its complexity score), but also on the experience level of the user, as well as its *presentation*. A user familiar with certain phenomena may deliberately search for such a phenomenon when trying to understand a justification, whereas a novice may not have this kind of coping strategy at their disposal. Regarding the presentation of a justification, the usual visual aids (e.g. in Protégé 4) when displaying OWL justifications on a screen are:

- Ordering: in simplified terms, given two axioms α and α', if α contains entities on the RHS which occur in the LHS of α', then α is placed before α' (assuming the axiom list is read left-to-right or top-to-bottom). All other

$A_3 \sqsubseteq A_4$
$A_5 \sqsubseteq A_6$
$A_2 \sqsubseteq A_3$
$A_4 \sqsubseteq A_5$
$A_1 \sqsubseteq A_2$

(a) Unordered justification.

$A_1 \sqsubseteq A_2$
$A_2 \sqsubseteq A_3$
$A_3 \sqsubseteq A_4$
$A_4 \sqsubseteq A_5$
$A_5 \sqsubseteq A_6$

(b) Ordered and indented justification.

Figure 6.1: Different representations of a justification for $A_1 \sqsubseteq A_6$.

axioms are placed at the end. Overall, the justification axioms are ordered to contain the RHS of the entailment at the beginning, and the LHS of the entailment at the end.

- Indentation: assuming that α' follows after α in a list of axioms which is ordered according to the above principles, α' is indented. Axioms which do not have any predecessors that share expressions are not indented.
- Syntax highlighting: keywords in the OWL Manchester syntax are highlighted in colour to stand out from the entity names.

There have been no direct investigations into the effects of these visual aids on understanding OWL justifications. However, in the exploratory study which formed the basis for the complexity model, Horridge [Hor11a] found that users often skip steps in atomic subsumption chains, a technique which is greatly supported by ordering and indentation. As an example, consider an atomic subsumption chain as shown in the example in Figure 6.1a and 6.1b.

If the chain is unordered and not indented, the user will have to search through the chain, jumping back and forth between the axioms, and *memorise* which concept they are looking for. While not *complex*, such a justification could still be considered challenging, whereby the challenge lies with the general *difficulty* of navigating in a non-linear fashion (see, for example, [JL80] for a discussion of the effects of sentence ordering on spatial reasoning). In contrast, if the chain is ordered and indented accordingly, a user—especially if familiar with this type of representation—will be able to recognise the chain structure and spot the entailment at a glance. The lack of ordering and indentation does not change the *reasoning* required to understand the justification, but it makes the justification as it is presented to the user more *difficult* to understand.

Furthermore, since syntax highlighting (as well as line indentation) has long been a default feature of programming environments, it seems an obvious step to apply it to human-readable OWL syntax. In summary, we can reasonably

assume that visual aids such as ordering, indentation and syntax highlighting improve the way users navigate through axioms, thus reducing the *difficulty* a user has in understanding a justification.

From complexity score to effort We have not yet answered the question of how to *measure* the effort a user has to apply in order to successfully understand and debug an entailment using a single justification. As there is no practical[2] way to measure the *cognitive effort* required by a person to solve a reasoning task, we have to base our measurements on more tangible metrics. Studies on human reasoning performance (e.g. [NBH+06]) commonly infer the complexity of a task from two values: the frequency of test subjects coming to an *erroneous* solution, and the *time* required to come to a solution.

Thus, we can assign an effort (or complexity) score c to each justification which is provided by some complexity model and corresponds to two factors:

1. The likelihood of the user finding an *unsuccesful* solution or not solving the debugging problem at all. An indicator for this is the number of incorrect modifications a user makes before finding a correct solution.
2. The time required to find a *successful* solution.

Regarding 2., previous experiments have given us some insights into how much time users spend with OWL justifications in general, and how much time they are willing to spend before giving up or declaring them 'impossible' to understand. Kalyanpur et al. [KPSH05, Kal06] conducted two studies in which OWL users were presented with several ontologies containing up to 30 unsatisfiable classes and were asked to repair these ontologies using different types of explanation tools. The average time a user spent per ontology was 12.9 minutes and 9.2 minutes, respectively, with a maximum of 33 minutes taken for one ontology. A further study conducted by Horridge et al. [HPS09] where users were asked to read and understand an arbitrary number of *individual* justifications showed that study subjects would choose to inspect an average of 18.9 justifications before ending the study. Finally, in a similar study [HBPS11a] involving a series of individual justifications, we found that test subjects would only spend an average of 2.5 minutes on a justification.

In summary, we can say that we have some vague indication of the time users

[2]This is not taking into account the possibility of measuring brain activity during problem solving using an fMRI procedure. See, for example, the prediction of brain activities using the ACT-R system: `http://act-r.psy.cmu.edu/actrnews/index.php?id=34`.

are willing to spend on understanding and repairing justifications. However, since all these studies were performed in an experimental setting, we do not know how the results translate into real-world situations where an ontology developer *must* fix a bug in an ontology and cannot simply give up. Some anecdotal evidence of the 'days and weeks' spent on debugging ontologies before the introduction of explanation into OWL tools can be found in [All05].

6.2.2 A model for user effort

Following on from the complexity of single justifications, it is straightforward to see that multiple justifications generally increase the effort a user has to put into debugging one or multiple entailments—if the justifications are considered in isolation, as we will assume in this effort model for dealing with multiple justifications. Recall our discussion of the number of justifications a user has to inspect in different scenarios of justification and entailment encounters (Section 6.1.2), which range from exactly 1 to the sum of all justifications for all entailments in a set of multiple entailments.

Now assume that each justification we encounter has a complexity score c which is provided by some complexity model and the times and error rates associated with the score. Not taking into account effects caused by learning (i.e. inspecting multiple justifications which contain a recurring pattern), fatigue, or representational issues (e.g. scrolling down a long list) when inspecting multiple justifications, we expect the total effort to be the sum of the individual justification effort measures. For example, if justification $\mathcal{J}_1$ requires the user 2 minutes to find a correct modification, and $\mathcal{J}_2$ requires 4 minutes, we expect the overall debugging process to require 6 minutes. Likewise, if a user is likely to solve $\mathcal{J}_1$ at the first attempt and requires 3 attempts at modifying $\mathcal{J}_2$, the overall number of incorrect modifications will be 4.

Based on these assumptions, we define our model for measuring user effort for a given debugging problem as a straightforward sum of the individual complexity scores for the justifications encountered: given a set of k entailments with j_i justifications each, the overall complexity is $\sum c_{ij}$ for $1 \leq i \leq k, 1 \leq j \leq j_i$, where c_{ij} denotes the complexity score of a justification $\mathcal{J}_{ij}$ for an entailment η_i. This is obviously only a worst case approximation, as we assume that a) the user deals with each justification individually, and b) the justifications are disjoint, that is, a modification of one justification does not affect any other justification.

However, since we do not yet have a sufficiently good understanding of how users interact with justifications and what strategies they commonly apply, this approximation will have to suffice for our purposes.

Reducing user effort Given this effort model, we can now also define the goal of 'reducing user effort': to exploit the justificatory structure of an ontology in order to reduce the overall time and error rates compared to those given by our effort model for encountering multiple justifications. This reduction may occur in two ways:

1. Introduce an *alleviation factor* $a_{ij} < 1.0$ that represents the reduction in complexity c_{ij} of a justification $\mathcal{J}_{ij}$ caused by some additional support provided to the user. The reduced complexity of $\mathcal{J}_{ij}$ is then $c_{ij} * a_{ij}$.
2. Lower the number of justifications a user has to inspect. The reduction of the number of justifications can be represented by an alleviation factor of $a_{ij} = 0$ for a justification $\mathcal{J}_{ij}$ that does not have to be 'touched' in the debugging process.

Note that for the purpose of defining the effort model, we assume that a user has to *understand* the justification in order to find a suitable repair; that is, the user has to inspect every axiom in the justification and apply mental reasoning strategies in order to understand how the axioms lead to the entailment. However, depending on the debugging strategy applied, the user may not have to inspect every axiom in order to find a suitable repair. Minimising the number of axioms encountered is a strategy mostly applied by semi-automated repair tools, such as the ontology revision tool proposed by Nikitina et al. [NRG12], in which users are presented with a series of axioms and have to make a simple accept/reject decision for each axiom. Reducing the number of axioms encountered is also one of the key aspects of the axiom ranking and the root and derived mechanism used in Kalyanpur's repair tool [Kal06]: by suggesting low-ranked axioms in (root) justifications for removal, the tool prevents the user from manually inspecting every axiom in a justification or set of justifications. In our effort model, a reduction in the number of axioms to inspect in a justification can also be represented by the alleviation factor a_{ij}, since fewer axioms to inspect can simply be regarded as a reduction in the overall complexity of the justification.

6.3 Coping strategies

Having defined how to measure a reduction in the user effort required to debug a set of justifications, we will now discuss a number of strategies which exploit the justificatory structure of an ontology in order to reduce the effort required by a user to debug one or multiple entailments. We will first outline the scenarios in which justifications can share axioms or be isomorphic, before moving on to suggestions of how the structural properties of a justification set can reduce user effort.

6.3.1 Characterising justification sets

In our analysis of justification encounters and debugging effort in the previous sections we have treated justifications as entirely independent entities, paying no attention to the structural relations between them. But we already know that there exists a variety of relations between them, such as overlap and structural isomorphism. Before proceeding with a discussion of potential interventions to mitigate the effects of multiple justifications, we will give an overview of the structural characteristics of justification sets. This will allow us to clearly identify those situations in which our proposed interventions will help, and those in which users need to fall back onto other strategies.

Given a set of multiple justifications, the set can have the following properties:

1. All justifications are disjoint and structurally different.
   (a) If the justifications are individually easy, there is no additional support required. While potentially tedious, users may eventually arrive at a suitable solution to the debugging task by dealing with the justifications one by one. If there are too many justifications, there is a risk of the user 'giving up' before all justifications are repaired.
   (b) In the worst case, the justifications are individually hard and there are no structural relations to take advantage of such that the justifications need to be tackled individually. Here, aids for understanding individual justifications (such as justification-based proofs or natural language explanations) may help reduce the complexity of the individual justifications.
   (c) If some justifications are easy and others hard, ranking them based on some complexity model and presenting the easiest justifications first

may decrease the likelihood of the user giving up immediately, while also taking advantage of learning effects over time. Further, complex justifications will benefit from the additional support mentioned in (b).

2. Some of the justifications overlap.
   (a) If all justifications overlap in some axioms and no two are disjoint, focus on the most frequent overlaps, e.g. by identifying an error in the shared axiom set, or by generating relevant lemmas to support users in understanding the overlapping subsets.
   (b) If there are some overlapping and some independent justifications, focus first on those which overlap, then continue as outlined in 1.
3. Some justifications are isomorphic.
   (a) Group isomorphic justifications in order to support the user in understanding the *template* first before tackling the individual justifications.
   (b) If there exist isomorphic justifications which also *overlap* in some axioms, apply both grouping and lemmatisation of the shared axiom set.
   (c) Treat independent justifications individually as outlined in 1.

6.3.2 Justification overlap

Single-axiom overlap Justification overlap containing only single axioms has been one of the key aspects of reducing debugging effort since the early days of justification-based explanation [SC03, Kal06]. Shared axioms are essential to finding minimal repairs, that is, minimal hitting sets across a set of justifications. The ontology editor Swoop uses the frequency of an axiom to compute a rank for the axiom, with high frequency axioms being recommended for removal, while the explanation tab in Protégé 4 displays the number of justification an axiom occurs in next to each axiom.

Single-axiom overlap can reduce user effort in several ways: first, indicating the frequency of an axiom *may* provide some hints towards a potentially erroneous axiom. If this is truly the case and an axiom occurring in $\sum j_i$ justifications turns out to be erroneous, the effort for successfully repairing the unwanted entailment is reduced to the effort required to understand and modify or remove this one axiom.

Second, if the high-frequency axiom itself is considered to be correct, it may still cause an error by interacting with other, incorrect axioms in the justification set. In this case, the number of justifications that need to be inspected remains

at $\sum j_i$, however, the number of *axioms* that the user encounters is lowered. This means that pointing out the shared axiom indirectly reduces the effort required to understand each one of the justifications, as the user is already 'familiar' with the shared axiom and only needs to understand how it interacts with the remaining axioms in each justification. Hence, we can introduce an alleviation factor a which indicates the reduction in effort required to understand the individual justifications. The overall effort is then reduced from $\sum c_{ij}$ to $\sum c_{i,j} * a_{ij}$ for $a_{ij} < 1.0$.

And third, the high-frequency axiom may only occur as a 'bridging' axiom which does not play a role in the actual conflict that leads to the unwanted entailment. This is the case, for example, in root and derived justifications for unsatisfiable classes, where the cause of the unsatisfiability of the classes lies within the root justification, whereas the remaining axioms in a derived justification simply 'bridge' the relationship between the classes. If the shared axiom is a bridging axiom, the reduction of effort is similar to the previous case: the effort required to understand each justification is reduced by a factor a_{ij} due to the reduction of the number of axioms the user encounters.

Justification equality The most striking effect of exploiting justificatory structure on the effort required for debugging a set of entailments is the case of justifications which are simply the same set of axioms. That is, breaking one justification instantly repairs all entailments in the set. Considering the user effort based on our simple effort model, given k entailments to debug with a single justification each, a user would have to inspect k justifications to debug all entailments. This is where the model is rather inaccurate, as we can assume that typical user behaviour would be to inspect one justification, apply a modification to it, then move on to the next justification, rather than inspecting all justifications first before applying a modification to one justification. If, after applying a modification, a reasoner is used to classify the ontology, the user would then immediately notice the repair effect on the other entailments. On the other hand, we can imagine that a user actively *tries* to find a minimal repair, thus inspecting all justifications first before applying any modification. In this case, the effort model still applies.

Regardless of the original effort to start with, it is straightforward to see that pointing out the equality between multiple justifications reduces the number of

justifications to inspect from $\sum j_i$ to 1. That is, the total effort required for the debugging task will be based on the complexity c_{ij} of that one justification.

Root and derived justifications Even if the justifications are not strictly equal, the existence of subset relationships can significantly reduce user effort in the debugging process. A straightforward application of subset relationships in a list-based justification representation is the *ordering* of justifications to present users with the *root* justifications first. Using a j-graph representation for a set of entailments and justifications, for example, the root justifications could simply be highlighted (for example by rendering them in colour) to signify the user that these justifications need to be dealt with first.

Regarding the reduction in user effort caused by root and derived justifications, consider a set of k entailments with a total of j justifications. Assume the set of justifications is partitioned into a set of root justifications $Justs_r$ and a set of derived justifications $Justs_d$, that is, $|Justs_r| + |Justs_d| = j$.

In the presence of root and derived justifications, the number of justifications a user has to repair in order to find a suitable repair for *all* entailments corresponds to the number of root justifications $|Justs_r|$. Given that $|Justs_d| > 0$, the number of justifications to inspect will always be lower than the initial number given by our model, with the reduction in effort depending on the ratio between root and derived justifications. As an effect, we can simply introduce an alleviation factor of $a_{ij} = 0$ for all *derived* justifications $\mathcal{J}_{ij}$.

Arbitrary overlap While the effects of justification equivalence and root and derived justifications are immediate, arbitrary justification overlap has a more indirect impact on debugging effort. As we have discussed in Section 4.3.3, arbitrary overlap can generate a relevant lemma, that is, an intermediate entailment of a subset of a justification. Such a common lemma which occurs in multiple justifications can support a user in *understanding* multiple justifications by prompting *chunking*, an effect commonly described in cognitive science [Mil56, GLC+01].

The restricted *working memory* of humans is known to be a limiting factor for a person's ability to process information. Cognitive science research widely agrees that the number of items a human can hold in their working memory at any time is fairly small, with figures commonly ranging from four items [HBMB05]

to 'the magical number seven' [Mil56].[3] The process of chunking, however, helps mitigate this limitation by grouping multiple items into one single *chunk*, which corresponds to a single item held in memory. Thus, chunking practically increases the number of items a human can deal with at the same time.

Lemmatisation of an overlapping justification subset corresponds to such chunking, as it groups a number of items—the individual axioms—into a chunk of which only the lemma is relevant to the user. This reduces the overall complexity of the justification set in two ways: first, the lemmatisation makes each individual justification easier to understand. And second, the complexity of understanding each justification is further reduced due to the lemma *reoccurring* in the justification set. Thus, while the number of justifications to inspect remains the same, the complexity of each justification is reduced by a factor a_{ij}.

6.3.3 Isomorphism relations

As we already know, the isomorphism relations introduced in the previous chapter are equivalence relations, that is, they *partition* a set of justifications into subsets of structurally similar justifications. This effect can be used to provide improved debugging support by *grouping* justifications into their equivalence classes and presenting the user with the abstract template for each group. Given a set of justifications for a single or multiple entailments, we first partition the justifications into the sets of isomorphic justifications according to some notion of isomorphism. These subsets are then arranged to show the user the template Θ to give an abstract explanation of the entire set of structurally similar justifications.

Similar to the exploitation of arbitrary overlap, such a grouping technique does not directly reduce the number of justifications to be inspected, but it reduces the overall complexity of the justification set the user is dealing with: by understanding the template first, we expect the user to spend less time and have less difficulty when repairing each individual justification. As we cannot apply a repair to a template (since it does not correspond directly to any material axioms in the ontology), the user will still have to repair each justification; the effort for this repair then depends on the overlap relations between the justifications. However, understanding the template first will reduce the overall effort from $\sum c_{ij}$ to

[3]Note that Miller's 'magical number' paper has been frequently mis-used to generalise to entirely unrelated problems such as text comprehension. See, for example, Edward Tufte's archive page [Tuf] on misinterpretations of Miller's paper.

$\sum c_{ij} * a_{ij}$, where $a_{ij} < 1.0$ for the justifications $\mathcal{J}_{ij}$ covered by the template.

Entity naming in justification templates In the examples in the previous chapter we used freshly introduced variable names for the abstractions from class, property, and individual names in a template. While this is straightforward to understand, there may be alternative ways of presenting these abstractions to a user which may be more suitable for understanding the similarity between the justifications. Given a set of classes (properties) in *strictly* isomorphic justifications which are mapped to a newly introduced variable x in a template Θ, the following strategies[4] can be used for naming x:

1. If the classes (properties) have a common named superclass C_s (superproperty p_s) use this to represent x; this superclass is also know as the *least common subsumer* [BKM99]. Alternatively, as the least common subsumer may be too general, use a *good common subsumer* [BST07], i.e. a superclass which is not the least common subsumer, but the most representative and informative for a user.
2. If there is no common superclass or superproperty, we can attempt to generate a new entity name by simply listing all entity names, e.g. as a comma separated list.

In the case of subexpression- and lemma-isomorphism, the approaches to naming abstract entities in Θ are less straightforward. For subexpression-isomorphism, we can attempt to find a common subsumer of the complex subexpressions that are mapped to a variable x. The situation is similar for lemma-isomorphic justifications: assume a set of justifications is lemma-isomorphic, that is, there exist lemmatisations of the justifications that are subexpression-isomorphic. Even in this case we can attempt to find common subsumers for the entities occurring in the lemmas to represent the entities in Θ. In either case, if there is no such common named subsumer, the naming has to default to using freshly generated variable names.

Levels of abstraction In the previous chapter, we established the notion of a preferred template Θ_p which is the smallest possible abstract representation of a set of isomorphic justifications. The type of isomorphism hereby determines

[4]The naming problem we are facing here is similar to that presented in Section 3.1.4 where we looked at suitable representations for subsumption relationships between equivalent class nodes in a class graph.

the level of abstraction of the preferred template: strict isomorphism results in a template which is structurally identical to the original justifications, whereas subexpression- and lemma-isomorphism abstract from the actual structure of the justification axioms, resulting in a more high-level view.

Note that there is no strict ordering of abstraction levels between subexpression-isomorphism and lemma-isomorphism: it is certainly possible for two justifications to be *nearly* strictly isomorphic with the exception of a subset S (such as a short atomic subsumption chain) in *one* of the justifications, such that the justifications are 'only' lemma-isomorphic. The template Θ_p will then, again, be structurally identical to the justification not containing S. In contrast, subexpression-isomorphism (on which lemma-isomorphism is based) may unite justifications that use a range of widely differing constructors; thus, the level of abstraction depends entirely on the numbers and types of transformations for each justification set, rather than the type of isomorphism.

In line with the idea behind laconic justifications—presenting users with as little unnecessary detail as possible—it seems reasonable to use only preferred templates in a user-facing application. However, which level of abstraction is suitable to best support a user in understanding and repairing a justification is unclear: a high-level, abstract representation of the justification (in some sense an 'explanation of the justification') may help the user grasp the basic reasoning behind the justification, which can be useful in getting them 'on the right path'.

On the other hand, when it comes to modifying the justification in order to remove the entailment, a detailed understanding of the expressions and axioms may be of more use, as this will help finding an appropriate modification. This was indeed one of the main concerns in the definition of *preferred* laconic justifications: to contain no superfluous parts, while being as close to the original justification as possible [HPS08]. However, what level of granularity for justification templates is appropriate in a debugging context remains an open question.

Combining isomorphism and overlap Finally, we can also combine isomorphism and justification overlap: it is possible for a set of justifications to be structurally isomorphic, i.e. differ within most of its entity names, but have a common axiom set which results in a relevant lemma. In this case, the user can be presented with the justification template Θ of the isomorphic justifications, whereby the common axioms are represented by their concrete lemma rather than

their abstract template.

Combining isomorphism and overlap to create lemmatised templates reduces the effort required to understand the set of justifications in two ways: first, the overall effort to understand all justifications is reduced due to the abstract template. Second, the complexity of the template is then further reduced by lemmatising it. That is, as previously, the overall effort is reduced from $\sum c_{ij}$ to $\sum c_{ij} * a_{ij}$ for $a_{ij} < 1.0$. However, due to the presence of a lemma, we expect the factor a_{ij} to be smaller compared to isomorphism without overlap.

6.4 Summary and conclusions

In this chapter, we have proposed different approaches to exploiting justificatory structure in order to reduce user effort in the ontology debugging process. We first defined a debugging problem as the problem of finding a minimum loss modification which can be applied to an ontology in order to remove all unwanted entailments while retaining as much wanted information as possible. Using this definition of a debugging problem, we introduced a model for measuring the effort required to solve a debugging problem, which is based on the number of justifications and axioms that need to be inspected in order to find a suitable modification. By introducing an alleviation factor a_{ij} to be applied to the complexity of an individual justification, we can then quantify the reduction in effort achieved by individual coping techniques.

We then introduced a number of coping strategies that make use of different aspects of justificatory structure, and outlined how these strategies would affect the effort required to repair single or multiple entailments. While we do not have any experimental results on the effectiveness of the proposed measures, these suggestions lay out the different paths that can be taken in order to improve justification-based debugging support by exploiting justificatory structure, while also proposing a way to *measure* the success of such strategies. While we have set the foundations for structure-based debugging support, it is clear to see that the successful application of such interventions in ontology tools will require further research into the way OWL developers read and understand justifications in order to determine suitable interaction mechanisms.

Chapter 7

A survey of justificatory structure

Following on from the introduction of various aspects of the justificatory structure of OWL ontologies and interventions for justification encounters in the previous chapters, this chapter presents a survey of the justificatory structure of a set of ontologies from the NCBO BioPortal, a curated collection of over 300 ontologies from the biomedical domain. Given the proposed structure-based coping strategies, we now want to know which structural aspects indeed occur in ontologies used in practice, and to what extent. While we have some knowledge about the occurrence of multiple justifications and some of the structural phenomena found in OWL justifications (e.g. [KPHS07, HBPS11a, BHPS11]), to date there has been no thorough investigation of the complex relationships between justifications in a large, independently motivated ontology corpus.

This chapter presents an investigation of the applicability of the theoretical concepts and interventions we have proposed in the previous chapters. The survey presented here indicates the clinical significance of both the suggested sources of difficulties, as well as the proposed structure-based interventions. In our case, we claim that the significance of phenomenon and intervention corresponds to its *prevalence* in OWL ontologies used in practice: given some OWL ontology, how likely is it for an ontology developer to encounter a source of difficulty, and how likely is it that structure-based coping strategies exist that can be exploited to reduce this difficulty?

7.1 The BioPortal corpus

In this section, we will describe the properties of the test corpus used in our survey, and outline the justification generation process. The entailment extraction and justification generation comprises several filtering steps, which are motivated by the different types of entailments justifications we have presented in Chapters

3 and 4: if we simply treated all justifications and entailments equally, regardless of their origin (native, mixed, or imported) or complexity (self-justification, atomic subsumption chain, or complex), we would run risk of over- or understating features of the justificatory structure of the ontologies in our corpus. We therefore filter out entailments—and ontologies—which would skew the justification analysis towards irrelevant types of justifications.

7.1.1 Properties of the corpus

The purpose of this survey is to analyse the justificatory structure of a set of OWL ontologies which is representative for the range of ontologies used in practise. This would exclude tutorial and toy ontologies, such as the well-known *Pizza* and *Koala* ontologies, which were specifically built for the purpose of demonstrating certain OWL features, or might be very small and inexpressive. In order to prevent bias through hand-picking 'suitable' ontologies, our aim was to use an independently motivated corpus of ontologies. Thus, the choice was between a random sample of web ontologies (e.g. obtained from a web crawl) and an existing ontology repository. As the NCBO BioPortal ontology repository contains a large number of actively used and well-studied ontologies which cover a broad spectrum of size and expressivity, it seemed a suitable choice for an extensive survey of justificatory structure.

BioPortal is a web-based ontology repository which provides over 300 ontologies published by research groups from the biomedical domain. It also includes the full set of OBO Foundry[1] ontologies, which use a flat-file format that can be translated into OWL 2; therefore, the OBO ontologies were included in the corpus. The activity of the repository varies throughout the ontologies: while some are updated (if only sporadically) and offer several prior versions, others have not been modified since their first upload. The repository has seen some visible growth in recent years, with the number of downloadable OWL and OBO ontologies increasing from 218 in March 2011 [BHPS11] to 256 in May 2012.

7.1.2 Justification corpus preparation

Ontology corpus At the time of downloading (May 2012), 322 OWL and OBO files were listed in BioPortal, of which 256 could be downloaded via the BioPortal

[1] `http://www.obofoundry.org/`

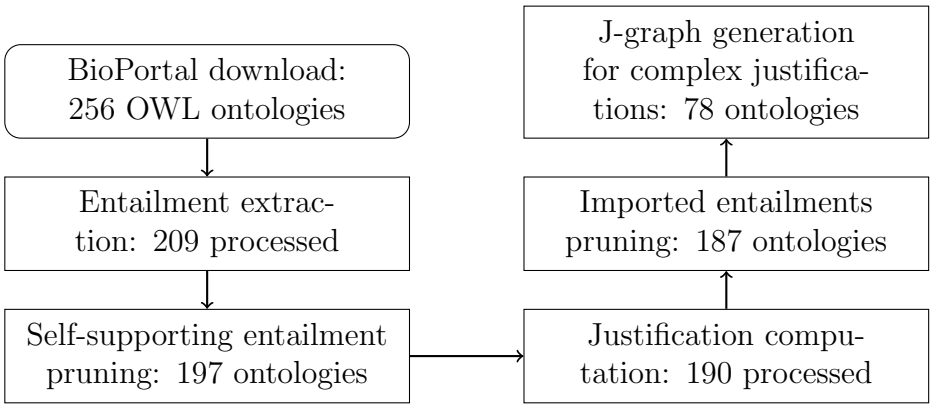


Figure 7.1: The justification corpus preparation workflow.

REST interface and successfully parsed by the OWL API. The main reasons for download failures of the remaining ontologies were 403 ('forbidden') server errors, parser problems caused by actual errors in the ontology file, and the ontology not being available at the given URL. Each downloadable ontology was merged with its imports closure and serialised as OWL/XML file. Imported axioms were being annotated with their source ontology URI, while missing imports were ignored. A diagram of the full corpus preparation workflow is shown in Figure 7.1.

Stage 1: entailment extraction Due to the small number of 'naturally occurring' unsatisfiable classes in published OWL ontologies, it seems reasonable to extend our analysis from obvious 'errors', that is, unsatisfiable classes, to include atomic subsumptions which represent the class hierarchy of an ontology. We computed entailment sets for the 256 downloaded ontologies including entailments of the following two types: 1) the $(ADI^{nim})^+$ entailment set of atomic subsumptions of type $A \sqsubseteq B$ for named, satisfiable classes A and B of each ontology, including self-supporting entailments, and excluding tautologies such as $A \sqsubseteq \top$, as well as all 2) the $(ADI^{nim})^+$ set of atomic subsumptions of the type $A \sqsubseteq \bot$ for unsatisfiable classes A in the ontology. The entailments were generated as follows:

1. Perform consistency check on the ontology.
2. Classify: `precomputeInferences(InferenceType.CLASS_HIERARCHY)`.
3. Extract entailed subsumption axioms using `InferredAxiomGenerator`.

For practical reasons, the processing times for the entailment generation was limited to a reasoner timeout (for consistency checking and classification) of 20 minutes. The overall timeout per ontology for each experiment in Stages 1 to

3 was set to 90 minutes. The reasoners used were JFact version 0.9 and Pellet version 2.3.0. All experiments were run on a Mac Mini with a 2.7 GHz Intel Core i7 processor, with 16 GB RAM assigned to the JVM.

In this stage, a total of 209 ontologies could be processed successfully using either Pellet or JFact, with the remaining 47 ontologies suffering from various errors such as inconsistency (7 ontologies), classification timeouts on both reasoners (9 ontologies), and `UnsupportedFeatureException` and `OutOfMemory` errors (13 ontologies) in the reasoners. 18 of these ontologies were discarded due to them containing no logical axioms, which was most likely caused by errors in the OWL/XML serialisation stage. For the remaining 209 ontologies, a total of 8,351,061 entailments were computed.

Stage 2: self-supporting entailments removal In order to limit the justification generation and analysis to an interesting set of entailments, we pruned all self-supporting entailments (type T_1, according to the classification introduced in Chapter 4), from the over 8 million entailments generated in Stage 1. This was simply done by removing the asserted axiom from the ontology, then performing an entailment check to see whether the axiom was still entailed, i.e. to see whether there were other non-self justifications. At this stage, 204 ontologies could be processed entirely without reasoner timeouts, whereas some of the ontologies contributing the most entailments timed out. Of the remaining 3,837,219 entailments, roughly one tenth (465,364) was found to have only self-justifications, and a total of 7 ontologies that contained only self-supporting entailments were discarded from the set. This resulted in 197 ontologies that contained 3,371,855 entailments that had *some* non-self justification.

Stage 3: justification generation and filtering In the next stage, the justifications were generated for the remaining entailments. The high runtime of the justification generation process made it necessary to limit the number of justifications generated to a reasonably large sample. Thus, the justification generation was restricted to a random sample of 1,000 entailments per ontology and 500 justifications per entailment, which, based on previous studies [BHPS11], was expected to yield a good balance between efficiency and size of the data set. The timeouts were set to 5 minutes per justification and 90 minutes per ontology.

In this stage, justifications could be generated for 115,675 entailments from 190 ontologies. 84 of these ontologies contained more than 1,000 entailments

in which case we generated the justifications for a random sample of 1,000 entailments. These data were then filtered to discard imported justifications, resulting in the removal of 4,937 purely imported entailments and 3 ontologies which contained only imported entailments. The final entailment set consisted of 317,774 native and mixed justifications for 110,738 entailments from 187 ontologies, including 419 unsatisfiable classes from 10 ontologies. For the purpose of analysing the corpus, it was split into the sets of *all* justifications for entailed atomic subsumptions involving satisfiable classes (S_{sa}) and those involving unsatisfiable classes (S_u). Table 7.1 shows an overview of the numbers of entailments, justifications, and ontologies in the two sets.

Table 7.1: Overview of data in sets S_{sa} and S_u.

	S_{sa}	S_u
Ontologies	187	10
Justifications	307,422	10,352
Entailments	110,319	419

Ontology properties The 187 ontologies in the corpus span a broad spectrum of sizes and expressivities, ranging from ontologies with only 4 classes and 5 logical axioms to as many as 12,195 classes and 79,180 axioms. Some of the basic metrics for the ontologies in the corpus are shown in Tables 7.3 and 7.2. A complete list of the relevant ontologies (i.e. those containing some complex justifications) including their metrics can be found in Appendix A.

Table 7.2: Overview of OWL 2 profiles.

Profile	Ontologies
OWL 2 Full	15
OWL 2 DL	162
OWL 2 EL	92
OWL 2 QL	53
OWL 2 RL	40

Regarding their expressivity, 162 of the 187 ontologies are OWL 2 DL ontologies, of which 92 are in the OWL 2 EL profile, 53 in OWL 2 QL, and 40 in OWL

2 RL. The remaining 15 ontologies were OWL 2 Full. Note that the three OWL 2 profiles are *not* exclusive, that is, an ontology can fall into several profiles. The description logic complexity in the corpus ranges from $\mathcal{EL}^{++}$ (the 92 ontologies in the OWL 2 EL profile) to full OWL 2 DL expressivity ($\mathcal{SROIQ}$, 5 ontologies), with 14 ontologies being in a highly expressive description logic ($\mathcal{SHQ}$ or $\mathcal{SRQ}$ including either inverse properties $\mathcal{I}$ or nominals $\mathcal{O}$).

Table 7.3: Overview of the basic ontology metrics in the corpus.

	Mean	Median	Min	Max
Classes	2,206.4	395	5	38,640
Object properties	22.2	6.5	0	431
Data properties	9.8	0	0	488
Individuals	159.6	0	0	7,559
Logical axioms	4856.6	810.5	19	79,180
Entailments (sampled)	608.8	710.5	1	1,000

7.2 Results of the BioPortal survey

In this section we will present the results of several experiments carried out on the BioPortal ontology corpus. The survey covers the most relevant aspects of justificatory structure we have introduced in the previous chapters: types and numbers of justifications, overlap (equality, root and derived, arbitrary overlap, axiom frequency) between justifications, and justification isomorphism. A discussion of the results follows in the next section.

7.2.1 Entailment types

In order to determine the distribution of entailment types in the corpus, the set of justifications was partitioned into self-justifications, atomic subsumption chains, and complex justifications, and their corresponding entailments. Table 7.4 shows the distribution of entailment types over the 110,319 entailments in S_{sa} and the 419 entailments in S_u. Note that a small number of entailments appears to be of type T_1 (as defined in Chapter 4) despite the previous removal of self-supporting entailments; this is caused by the fact that we could not compute additional justifications for these entailments (e.g. due to timeouts).

We can clearly see that the majority of entailments in S_{sa} (81.7%) has only atomic subsumption chain justifications. This also affects the majority of ontologies in the corpus: 109 out of the 187 ontologies for which we generated justifications in S_{sa} contained no complex justifications at all, but only atomic subsumption chain justifications and small numbers of self-justifications. While ranging in size from 5 to over 77,000 axioms, most of these ontologies are only of weak expressivity: 79 (72.4%) of the 109 ontologies that contain only trivial entailments and justifications are in the description logic $\mathcal{EL}^{++}$, 3 are in $\mathcal{SHI}$, $\mathcal{SHIF}$, and $\mathcal{SRIQ}$, respectively, and the remaining 27 ontologies are in variations of $\mathcal{AL}$.

All entailments involving unsatisfiable classes in S_u have only complex justifications. Presumably this is because self-justifications or atomic subsumption chain justifications for an unsatisfiable class would require an axiom of type $A \sqsubseteq \bot$ to be asserted in the ontology, which is unlikely.

Note that due to the small number of entailments and ontologies in S_u and the domination of one ontology (322 of the 419 entailments are contributed by the *Animal Natural History* ontology), the results for S_u in this section are mainly given for completeness where appropriate.

Table 7.4: Entailment types in sets S_{sa} and S_u.

Type	Description	S_{sa}	S_u
T_1	self-justifications only	23	0
T_2	atomic subsumption chains only	90,460	0
T_3	self-justifications and atomic subs. chains	74	0
T_4	complex justifications only	4,983	419
T_5	self-justifications and complex	1,042	0
T_6	atomic subsumption chains and complex	13,551	0
T_7	all justification types	186	0
	total entailments	110,319	419

7.2.2 Occurrence of multiple justifications

One of the main issues we have set out to explore in this thesis is the occurrence of multiple justifications in OWL ontologies: given an entailment of an ontology, how likely is it that this entailment has several justifications, and what are the

chances of encountering an entailment with a high number of justifications? We generated the j-graphs for the *complex* justifications of the entailments of types T_4 through T_7 in sets S_{sa} and S_u. The average time taken to compute each j-graph using the existing justifications was less than 10 seconds. This resulted in j-graphs for a total of 145,689 complex justifications for 19,772 entailments from 78 ontologies in set S_{sa}. In the remainder of this section, we will use S_s to refer to the set of *complex* justifications in S_{sa}. Note that the justification count indicates the number of justification *nodes* in the j-graph, which takes into account the fact that some justifications have multiple entailments.

The average number of (complex) justifications per entailment in S_s is surprisingly high, with 7.8 justifications (standard deviation $\sigma = 26.1$, median $m = 3$) and a maximum of 500 generated justifications for 17 entailments in 4 ontologies. Figure 7.2 shows the frequency of multiple justifications for the entailments and ontologies in the S_s.[2] Approximately one third (30.0%) of the entailments in the corpus have exactly one justification, whereas 18.1% have exactly 2 justifications; half of the entailments (52%) have 3 or more justifications, and a significant proportion (15.7%) of the entailments reach 10 or more justifications.

The ontology frequency plot in Figure 7.2 shows that these numbers are not only caused by single ontologies which happen to have unusually large numbers of justifications. 69 out of the 78 ontologies (88.5%) in the set contain entailments with 2 or more justifications, 58 ontologies have entailments with 3 or more justifications, and 19 ontologies contain some entailment with 10 or more justifications. Interestingly, 8 ontologies in the set contain *only* multiple justifications for their entailments.

In S_u, the average number of justifications per entailment is 24.1 ($\sigma = 12.8$, $m = 32$), with a maximum of 55 justifications for entailments in the *Quantitative Imaging Biomarker* ontology. 7 of the 10 ontologies contain only entailments with exactly 1 justification, whereas the remaining 3 ontologies contain only entailments with multiple justifications.

While we might expect to see a correlation between the size of an ontology and the number of complex justification it generates, we found that there are no obvious indicators for the occurrence of multiple justifications in an ontology. The Spearman rank coefficient[3] $\rho = 0.09$ ($p = 0.43$) indicates no correlation

[2]Note that the 'long tail' of the plot has been cut off for presentation purposes.

[3]A coefficient ρ of +1 (-1) indicates a strong positive (negative) correlation between two variables, whereas a ρ of 0 indicates no correlation.

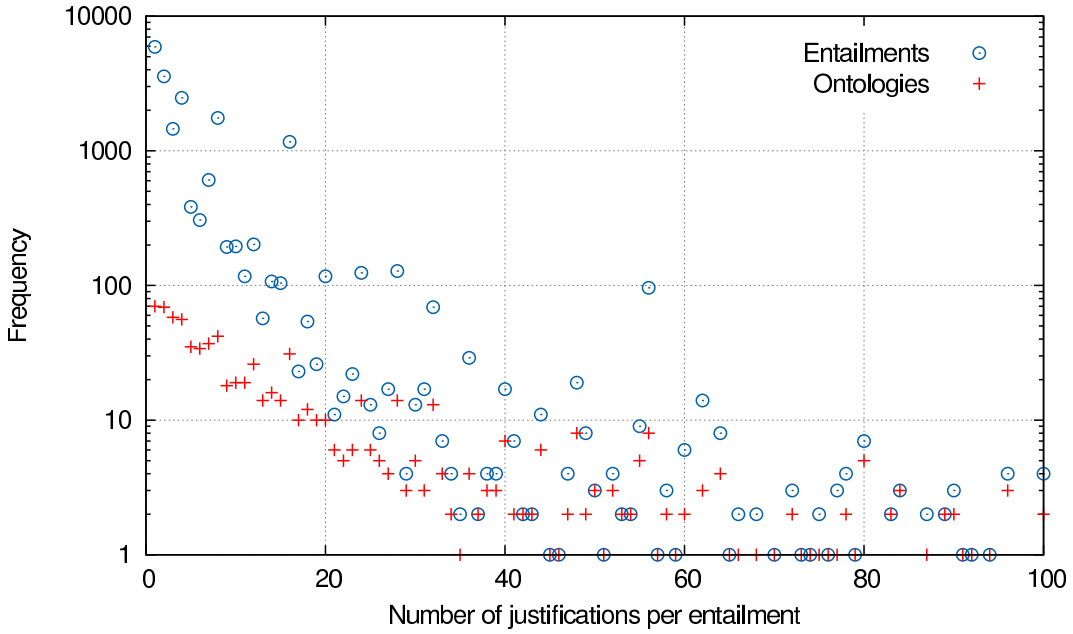


Figure 7.2: Frequency of multiple complex justifications in the corpus.

between the number of logical axioms in an ontology in set S_s and the number of justifications per entailment. Due to the small number of ontologies in S_u, the correlation analysis was only performed for S_s.

Ontology activity and axiom power The 145,689 complex justifications in S_s contain a total of 22,371 axioms. On average, one fifth of the axioms in an ontology (22.2%, $\sigma = 20.4\%$, $m = 15.1\%$) are active in complex justifications for their entailed atomic subsumptions, with some ontologies having as many as 75.5% of their axioms participating in justifications. Note that the small number of axioms compared to the number of justifications indicates a high amount of shared axioms in the corpus; we will focus on the matter of axiom overlap below.

Intuitively, we would expect to see an obvious correlation between the number of entailments in an ontology and its activity (the proportion of active axioms), since more entailments would imply more justifications and axioms occurring in them; this is somewhat confirmed by $\rho = 0.49$ (Pearson's correlation coefficient) indicating a weak linear correlation ($p < 0.001$).

Graph components In order to determine the overall connectedness of justifications, we consider the number of connected components in each j-graph. On

average, each graph contains 4.6 (σ =9.8, m =2) connected components, but almost half of the graphs in S_s (37 out of 78) consist of exactly one component. The largest number of components can be found in the *International Classification for Nursing Practice* ontology, which has 1,063 complex justifications for 762 entailments that are split up over 68 components.

As we have mentioned in previous chapters, the justification finding algorithm uses the Hitting Set Tree algorithm, which depends on optimisations such as justification reuse. This implies that largely disjoint justification sets may have a negative impact on the performance of the 'find all' algorithm. However, there seems to be no correlation between the number of components in a j-graph and the average time required to find a justification ($\rho = 0.13$, $p = 0.25$), or the number of components and the number of calls to the 'find one' subroutine ($\rho = 0.06$, $p = 0.58$).[4]

7.2.3 Justification overlap

Inferential power of justifications While multiple justifications per entailment occur very frequently in the corpus, the number of justifications with multiple entailments is comparatively low, with only 709 justifications in S_s having out-degrees ≥ 2 in the j-graph. On average, a justification in S_s has 1.1 entailments ($\sigma = 2.2$, $m = 1$), with a maximum of 120 entailments for 16 fairly small justifications (containing 5 to 7 axioms) in the *SNP* ontology. Across the corpus, only 17 out of the 78 ontologies contain justifications with multiple entailments.

There is no correlation between the size of a justification and its number of entailments (ρ = -0.07, $p < 0.001$). The average size of justifications that have only one entailment is 7.9, with some of the largest justifications (20 and more axioms) having only one entailment, whereas the average size of a justification with multiple entailments is 4.2 axioms.

Axiom frequency and impact Regarding the frequency and impact of the axioms in the corpus, we want to find out how frequently axioms occur in multiple justifications, and how many entailments these axioms affect. On average, an axiom occurs in 51.3 justifications ($\sigma = 242.1$, $m = 5$) in set S_s, with some key axioms in several ontologies occurring in thousands of justifications. Figure

[4]Note, however, the high p-values which indicate that these findings are not statistically significant.

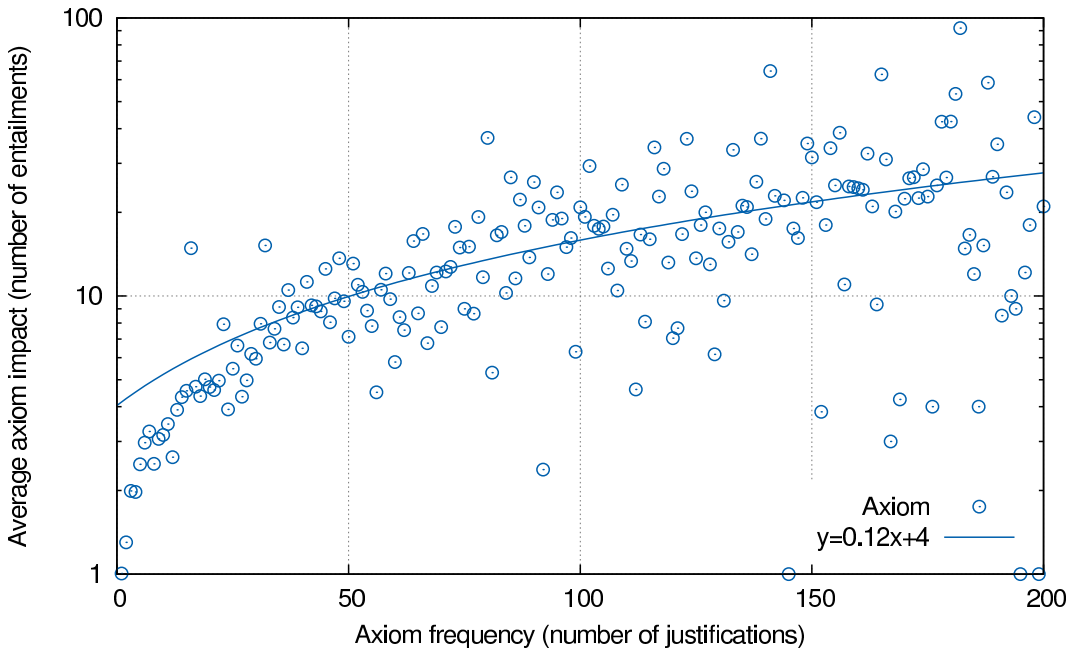


Figure 7.3: Axiom frequency vs average axiom impact, fitted with a linear trendline.

7.3 shows the axiom frequency plotted against the average impact, following approximately a linear distribution as shown by the linear trendline.[5]

The closeness of axiom frequency and impact is not surprising, since we have already seen that most justifications in the corpus only have a single entailment; therefore, the impact for most axioms is less than (if they occur in justifications for the same entailment) or equal to (if they occur in justifications for different entailments) the number of justifications they occur in.

Interestingly, all the ontologies in the corpus (even those which contain only small numbers of justifications) contain some axiom which occurs in multiple justifications, and 23 ontologies contain some axiom which occurs in over 50% of the justifications in the ontology. On average, the one axiom with the highest frequency in an ontology occurs in 36.3% of all justifications of an ontology.

An example of such a high-frequency axiom is the simple atomic subsumption $\mathsf{Disease} \sqsubseteq \mathsf{Disposition}$ which is used in 3,119 (96.5%) of the 3,232 justifications in the *NIF Dysfunction* ontology and affects 423 entailments. Another example is the domain axiom $\mathsf{domain}(\mathsf{measuredBy}, \mathsf{MentalConcept})$ in the *Cognitive Atlas* ontology, which occurs in 92.5% of the 830 justifications for 126 entailments.

[5]Again, the 'long tail' of the plot of up to 7,343 justifications has been cut off for presentation purposes.

Table 7.5: Root and derived justifications in S_s and S_u, in number of justifications and proportion of the total.

		$Justs_{rsub}$		$Justs_d$		$Justs_r$	
Set	Total	Count	%	Count	%	Count	%
S_s	145,689	10,723	7.4%	107,002	73.4%	27,964	19.2%
S_u	10,352	115	1.1%	10,184	98.4%	53	0.5%

Root and derived justifications Root and derived relationships occur frequently across the corpus: 73.4% of the justifications in S_s are derived, and 7.4% of the justifications are root justifications which have derived justifications. The remaining 19.2% are root justifications that do not have any justifications that are derived from them, that is, they are either additional independent justifications for derived entailments, or justifications for entirely independent root entailments. Table 7.5 gives an overview of the root and derived relationships in the corpus; root justifications that are subsets of derived justifications are denoted by $Justs_{rsub}$, root justifications that are no subsets are denoted by $Justs_r$, and derived justifications are denoted by $Justs_d$.

Looking at the repair impact of root justifications, we find that a justification in $Justs_{rsub}$ has 17 justifications on average which are derived from it ($\sigma = 92.8$, $m = 3$), and 5.4 entailments ($\sigma = 19.5$, $m = 2$) that are entailed by these derived justifications. In other words, while a root justification may have a fairly large number of justifications (17) that are derived from it, the number of entailments that can be repaired by fixing a single root justification is comparatively low (5.4).

The subset relationship between root and derived justifications is also visible in the size of justifications: on average, a root justification contains 4.7 axioms, whereas a derived justification has a size of 8.4 axioms. 7.6% of the root justifications are indeed single axiom justifications which occur in a large number of derived justifications. However, while we may expect a small justification to be more likely to be contained in derived justifications, there is only a weak correlation between the size of a root justification and the number of justifications that are derived from it ($\rho = -0.26, p < 0.001$).

70 of the 78 ontologies in S_s contain some derived justifications. 6 of of the 8 remaining ontologies only contain very small numbers of justifications, which

do not lend themselves to subset relationships, whereas the *Cancer Research and Management* ontology and *Gene Ontology Extension* contain no derived justifications despite having 348 and 6,421 justifications, respectively.

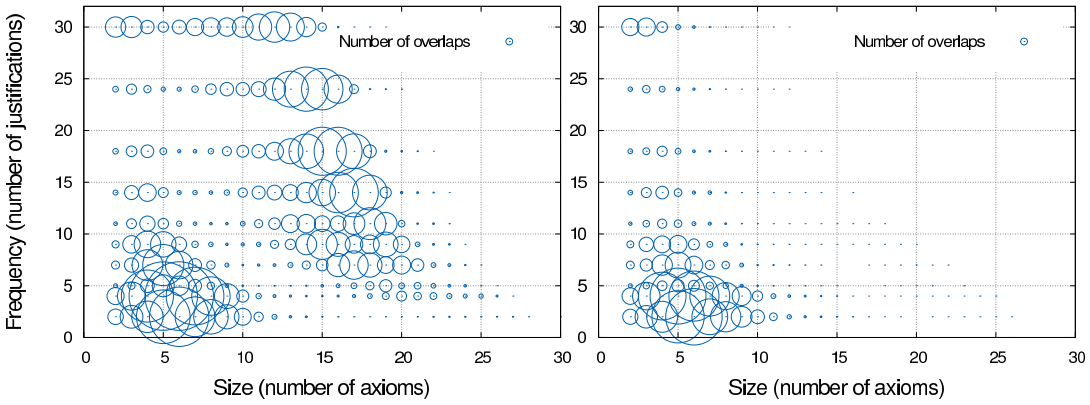
In the set S_u, the effect of root and derived justifications is even more pronounced: 98.4% of the justifications are derived from only 1.1% of the justifications, and the remaining 0.5% are root justifications which have no derived justifications. On average, a root justification has 88.6 derived justifications (σ = 208.6, $m = 2$) which have 48.8 entailments ($\sigma = 109.3$, $m = 2$). Note how this stands in contrast to the rather small number of entailments (5.4) that depend on root justifications in S_s. 5 of the 10 ontologies in S_u contain root and derived unsatisfiable classes, whereas the remaining 5 ontologies contain only very few unsatisfiable classes (up to 4) at all.

Arbitrary overlap In order to determine the numbers and sizes of justification overlaps with more than a single axiom, we applied the Formal Concept Analysis (FCA) [GSW05] 'next concept' algorithm to the j-graphs in sets S_s and S_u. This algorithm (often simply referred to as 'Ganter's algorithm') provides an efficient means for computing the largest shared axiom sets between justifications. The ToscanaJ FCA framework[6] includes a straight-forward implementation of Ganter's algorithm, which we used for overlap detection. Using the FCA algorithm, the justifications correspond to the *objects* in a context, and the axioms correspond to the *attributes*. Due to performance issues, the experiment was restricted to a random sample of maximum 5,000 edges per graph.

Across the ontologies in S_s, overlaps between justifications occur frequently, with an average size of 10.8 shared axioms ($\sigma = 5.7$, $m = 11$) and an average frequency of 11.9 justifications ($\sigma = 11.8$, $m = 9$) that the shared axiom set occurs in, i.e. on average, 11.9 justifications share the same axiom set. Figures 7.4a and 7.4b show the frequency of overlap of different size and frequency across the corpus including and excluding the two ontologies which contribute the largest numbers of overlaps. For presentation purposes, the values on the y-axis (i.e. the number of justification an overlap of a certain size occurs in) were binned in 10th percentile steps, leading to all overlaps with a frequency greater than 24 falling into the top-most bin.

The plot shows that the majority of overlaps (i.e. the largest bubbles) have a

[6] `http://toscanaj.sourceforge.net/`



(a) Overlap frequency for **all** ontologies in S_s.

(b) Overlap frequency **excluding** *Amino Acid* and *Basic Vertebrate Anatomy*.

Figure 7.4: Overlap frequency with and without outlier ontologies. Bubble size indicates frequency.

size of around 5 axioms and occur in up to 5 justifications. However, there are also overlapping cores of around 15 to 20 axioms which occur fairly frequently in up to 18 justifications; as we can see if we compare 7.4a to 7.4b, these large, high-frequency overlaps are almost exclusively contributed by the *Amino Acid* and *Basic Vertebrate Anatomy* ontologies. If we exclude these two ontologies from the set, the overlaps are reduced to an average of 5.5 axioms ($\sigma = 2.5$, $m = 5$) and a frequency of 7.8 justifications ($\sigma = 14.3$, $m = 4$).

Nearly all ontologies in S_s (75 out of 78) contain at least some overlapping justification subset, with two ontologies standing out as extreme outliers. The *Amino Acid* ontology contributes over half of the found overlaps in the corpus, which is rather surprising, as with 112 entailments, 477 axioms, and a description logic expressivity of $\mathcal{ALCF}$, this ontology could be considered fairly inexpressive.

However, if we take a closer look at the *Amino Acid* ontology, we find that it contains an axiom of the type $\mathsf{AminoAcid} \equiv \mathsf{A} \sqcup \mathsf{C} \sqcup \mathsf{D} \ldots \sqcup \mathsf{Y}$ with 20 named classes in the disjunction. This axiom, in turn, leads to a large number of justifications (2,652) containing subsumption axioms for each operand in the disjunction, plus a small number of 'bridging' axioms that cause the entailment to hold. For 1,782 of the 2,562 justifications, this axiom pulls in a large core of up to 29 other axioms, thus leading to the large numbers of overlaps observed in the ontology.

A similar effect can be seen in the *Basic Vertebrate Anatomy* ontology, another fairly small ontology (99 classes, 386 axioms, $\mathcal{SHIF}$) which, however, has a high number of object properties (74 compared an average of 22.2). The ontology

contains a number of axioms describing the relations between its object properties as part-whole relationships, such as $\mathsf{has\_subdivision} \sqsubseteq \mathsf{has\_determinate\_part}$ and $\mathsf{is\_subdivision\_of} \equiv \mathsf{has\_subdivision}^{-}$. Again, these property axioms pull in other 'bridging' axioms which leads to cores of up to 17 axioms occurring in large numbers of justifications.

Both of these outlier ontologies have high degrees of overlaps between multiple justifications for *single* entailments, but only small numbers of overlaps between justifications for several different entailments. This agrees with the general trend in the corpus, where on average those justifications which do share an overlap only have between 1 and 2 entailments (mean = 1.9, $m = 1$, $\sigma = 5.1$). In other words, overlap tends to occur mainly between justifications for the *same* entailment.

7.2.4 Justification isomorphism

In order to determine the frequency of isomorphic justifications in the corpus, we analysed the complex justifications for the entailments in set S_s and S_u in three different experiments:

1. justifications for a *single* entailment.
2. justifications for *all* entailments within an ontology.
3. justifications for *all* entailments of all ontologies across the corpus.

The isomorphism statistics were generated for all three types of isomorphism, that is, strict, subexpression-, and lemma-isomorphism.

Due to performance issues, justifications containing more than 10 axioms or conjunctions/disjunctions with more than 5 operands were excluded from this part of the study. This excluded some dominant justifications (such as those containing the large disjunction axiom in the *Amino Acid* ontology mentioned above), but still resulted in a set of 141,560 justifications for 19,097 entailments in S_s, which is an average of 7.4 justifications per entailment (compared to 7.8). Note that this also means that the total numbers of justifications given in this section for some of the ontologies will differ from the number of complex justifications listed in the overview table in Appendix A.

The mean times required to determine isomorphism between justifications for each of the three types (including parsing the justifications and existing templates from file) are listed in Table 7.6. Note that the experiments for the three isomorphism types were run in parallel, which could potentially cause a drop

or fluctuations in performance (as the 'Within ontology' times for strict versus subexpression-isomorphism show). Furthermore, the checks within ontologies and across the corpus each reuse the templates found in the previous stage, which drastically reduces the number of comparisons that have to be carried out.

Table 7.6: Mean times (in seconds) per ontology for isomorphism detection.

Size	Iso	S-iso	L-iso
Individual entailments	22.8	48.5	100.8
Within ontology	103.8	99.2	192.3
Across corpus	24.6	42.3	97.7

Individual entailments

Strict isomorphism Due to restrictions on the types of justifications used in the isomorphism experiments, the number of justifications for the entailments in this subset of S_s is marginally lower than in the j-graph analysis, with an average of 7.4 justifications per entailment. Strict isomorphism reduces this number to an average of 4.9 templates per entailment ($\sigma = 9.5$, $m = 2$), which is a reduction by 33.7% compared to the full corpus.

On average, a template covers 1.5 justifications ($\sigma = 2.3$, $m = 1$), with some ontologies containing entailments with large numbers of isomorphic justifications. One such example is the *Orphanet Ontology of Rare Diseases*, whose dominating templates are of the type

$$\Theta_1 = \{C1 \sqsubseteq C2, C2 \sqsubseteq \exists p1.C4, domain(p1, C3)\} \models C1 \sqsubseteq C3$$

with atomic subsumption chains of arbitrary size in place of the first subsumption axiom, and some variations that include subproperty axioms. Two of the templates of this type cover the majority (110 and 105 justifications, respectively) of the 220 justifications each for several entailments in the ontology. From personal contact with the *Orphanet* developers we have learned that this OWL ontology is in fact generated automatically from an existing medical database, which explains the frequent occurrences of uniform justifications.

Subexpression-isomorphism Subexpression-isomorphism across the justifications of individual entailments only affects a very small number of entailments

in the corpus. The average number of templates per entailment, compared to the full corpus, remains the same when rounded,[7] at 4.9 templates per entailment ($\sigma = 9.5$, $m = 2$). Compared to strict isomorphism, the number of templates is reduced by only 0.3%. This small reduction does not affect the average number of justifications per entailment, which remains the same at 1.5 justifications.

Only 166 entailments (0.9% of the total corpus) from 15 of the 78 ontologies are affected by subexpression-isomorphism; the remaining 63 ontologies contain the same number of templates as for strict isomorphism. For those ontologies that are affected, the decrease in templates compared to strict isomorphism is an average of 19.8%, reaching up to 50% for 9 entailments in the *Bleeding History Phenotype* ontology.

Lemma-isomorphism While subexpression-isomorphism does not have a strong effect on individual entailments, lemma-isomorphism shows a more visible reduction in template numbers.[8] On average, the justifications are reduced to 4.7 templates per entailment ($\sigma = 8.8$, $m = 2$), which is reduction by 4.1% compared to both strict and subexpression-isomorphism.

A total of 1,492 entailments (7.8% of the total corpus) from 43 ontologies are affected by lemma-isomorphism, with an average reduction of 30.3% compared to strict isomorphism for those entailments. The strongest effects can be seen in the *Fission Yeast Phenotype* ontology, where the justifications for several entailments only differ in the length of their atomic subsumption chains and thus are each reduced to a single template of the type

$$\Theta_2 = \{C1 \sqsubseteq \ldots \sqsubseteq C_n, C_n \equiv C2 \sqcap \ldots\} \models C1 \sqsubseteq C2.$$

Isomorphism within ontologies

Across the justifications for all entailments of an ontology, the reductions caused by the three equivalence relations are more clearly visible than for individual entailments. Figure 7.5 shows an overview of the effects of the different isomorphism types for the 25 ontologies containing the largest numbers of justifications in the corpus. Each cluster represents an ontology, with the ontologies ordered by the

[7] The precise number of templates is 4.915 for strict and 4.903 for subexpression-isomorphism.

[8] Note that S_s and S_u do not contain any justifications that consist entirely of atomic subsumption chains. That is, all atomic subsumption chains that are affected by l-isomorphism will be strict subsets of the complex justifications in the corpus.

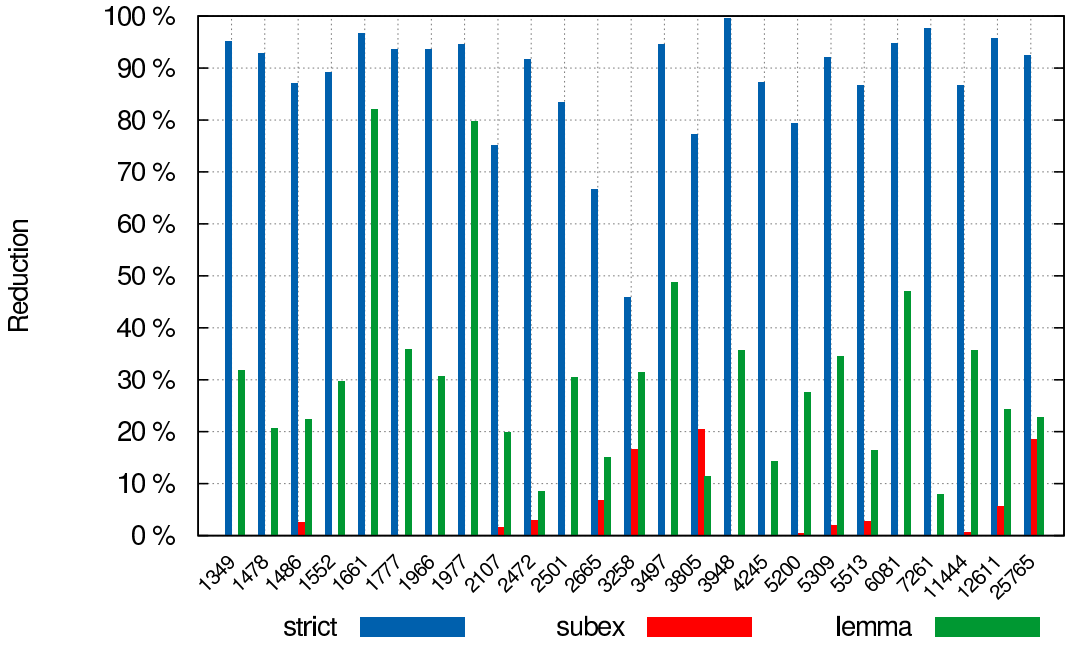


Figure 7.5: Comparison of reduction caused by isomorphism types. Each cluster represents an ontology. Ontologies are ordered by number of justifications.

number of justifications they contain. Each bar in a cluster represents the reduction compared to the previous equivalence relation (that is, we compare strict isomorphism with the full set of justifications, s-isomorphism with strict isomorphism, and l-isomorphism with s-isomorphism). We can see that the effects of the relations differ strongly across the ontologies, with strict isomorphism generally having the strongest impact, and subexpression-isomorphism having the lowest impact.

Overall, only three ontologies show no effect for *any* of the three types of isomorphism; two of these contain only a single justification (which means there is no reduction possible), while one ontology (*OBOE SBC*) contains 5 distinct justifications.

Strict isomorphism Compared to the results for individual entailments, the effects of the equivalence relations are much more significant if we consider the justifications for *all* entailments in an ontology. On average, an ontology in S_s contains 1,814.9 justifications; these are reduced to 436.1 templates through strict isomorphism, which is an average reduction by 73.1%.

4 ontologies with small numbers of justifications (less than 10) are not affected by strict isomorphism, while the remaining 74 ontologies show reductions of up

to 99.6%. Even more strikingly, 28 of the ontologies are reduced to less than 10% of their justification corpus. With an average number of 3,092 justifications per ontology, these reductions reveal significant numbers of structurally similar justifications.

The templates cover an average of 8.2 justifications each ($\sigma = 22.4$, $m = 2$), with some of the highest coverage occurring in the *Orphanet* ontology. The 3,948 justifications in this ontology can be reduced to only 14 templates, which are all variations (featuring atomic subsumption chains of varying length) of the template Θ_2 we have shown above.

Subexpression-Isomorphism Subexpression-isomorphism reduces the justifications in the corpus by an average of 74% per ontology, which is only a small change (0.9%) compared to strict isomorphism. The number of justifications covered by a single template is slightly increased to 8.8 ($\sigma = 23.5$, $m = 2$) justifications per template.

Again, the majority of ontologies (44 of 78) are not affected by subexpression-isomorphism, whereas 5 ontologies show reductions of between 20% and 38.3% compared to strict isomorphism. This includes the *Bleeding History Phenotype* ontology (1,158 justifications, 60 isomorphism templates, 37 s-isomorphism templates), which contains a number of justifications of the type

$$\{C1 \sqsubseteq \exists p1.(C2 \sqcup C3), domain(p1, C4)\} \models C1 \sqsubseteq C4$$

which is subexpression-isomorphic to justifications which contain a named class in place of the disjunction $C2 \sqcup C3$ in the subsumption axiom, thus matching template Θ_1.

Closer inspection of the *Lipid* ontology (3,258 justifications as shown in Figure 7.5, 1,762 isomorphism templates, 1,471 s-isomorphism templates) reveals that a large number of justifications in the ontology consist of a single equivalence axiom of the form $C1 \sqsubseteq C2 \sqcap x$ with the entailment being $C1 \sqsubseteq C2$. The remainder of the conjunction, represented by x, consists of a number of complex expressions of varying length and nesting depth. While s-isomorphism captures these types of justifications (since the remainders x can all be matched against each other), the actual reason for their similarity lies in their identical *cores* $C1 \sqsubseteq C2$, with the remainder x being a superfluous part.

Table 7.7: Template frequency and coverage across the corpus.

			Coverage			
Type	Count	% of S_s	Mean	Median	Min	Max
all	141,560	100%	-	-	-	-
strict	12,527	8.8%	11.3	2	1	2,072
subex	10,952	7.8%	12.9	2	1	2,128
lemma	5,487	3.1%	25.8	3	1	7,490

Lemma-isomorphism As for single entailments, lemma-isomorphism has a more significant effect on the justification corpus than subexpression-isomorphism when applied across each ontology. L-isomorphism reduces the justifications in an ontology by an average of 78.2%, which is a 4.2% difference compared to subexpression-isomorphism.

A template covers an average of 12 justifications ($\sigma = 38$, $m = 3$) in each ontology, which is a visible increase from the 8.8 justifications covered by subexpression-isomorphism. 13 ontologies (with an average number of 32.1 justifications) show no reaction to l-isomorphism, whereas 12 ontologies containing large numbers of justifications (mean = 1,555.8) see a reduction of 41.7% and more, up to 82.1% compared to subexpression-isomorphism.

Isomorphism across the corpus

In the final stage of our analysis of isomorphism in the BioPortal corpus, we will look at the templates spanning all justifications for all entailments across the ontologies in the corpus. While we have seen that isomorphic justifications occur frequently within an ontology, we now analyse the structural similarity of justifications across multiple ontologies. Table 7.7 gives an overview of the numbers of templates in S_s for the three isomorphism types against the full justification corpus, as well as the coverage (number of justifications) per template.

Strict isomorphism The 141,560 justifications in S_s are reduced to only 12,527 templates, which is a reduction by 91.2%. On average, 11.3 justifications ($\sigma = 54.3$, $m = 2$) share the same template, with the 8 most frequent templates covering over 1,000 justifications each. The most frequent templates (by numbers of justifications covered) are all variations of the template Θ_1, that is, a combination

of a domain axiom and a subsumption axiom with an existential restriction on the RHS, including some additional subsumption axioms, an equivalence axiom in place of a subsumption, or variations of the filler.

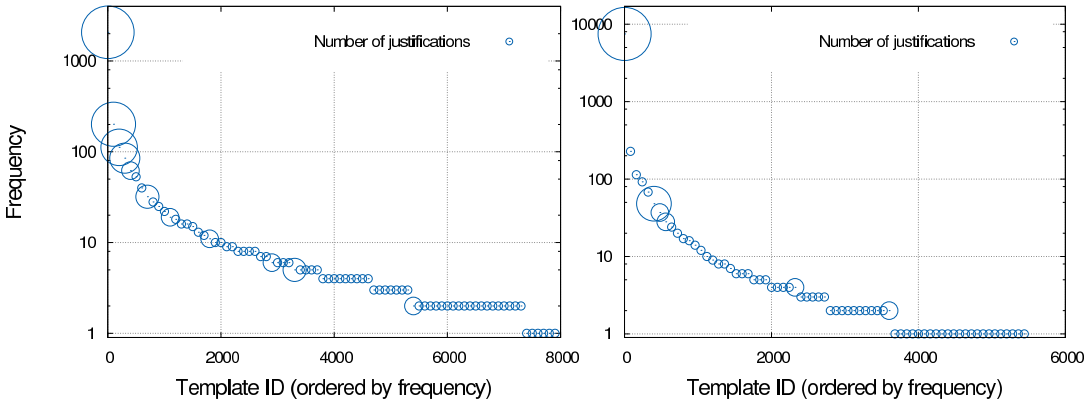
Nearly half of all templates (41.3%) in the corpus cover exactly one justification in a single ontology. We might larger templates to be more 'specific' to a particular ontology, thus covering fewer justifications overall, and conversely, a smaller template might be more 'generic', thus covering more justifications. However, the size (number of axioms) of a template seems to be no indicator for the number of justifications it covers: the average size (7.3 axioms) of a template covering multiple justifications is only marginally lower than of those templates covering a single justification (7.5 axioms).

If we look at the spread of templates across the ontologies in the corpus, we find that only 8.7% of the templates cover justifications in multiple ontologies, with a maximum spread of 26 ontologies for two templates which are, again, variations of Θ_1. Figure 7.6a shows the frequency distributions of the justification templates across the justifications and ontologies in S_s. The y-position of a data point indicates the number of justifications a template covers is, and the bubble size indicates the number of ontologies a template occurs in.

We can see that a small number of templates covers large numbers of justifications, with a steep drop to 10 and less justifications per template around the 2,000 mark. If we take a closer look at the numbers, we find that the majority of justifications in the corpus (81.2%) are covered by the 2,000 most frequent templates (out of 12,527), and a third (33.1%) of justifications are even covered by the 100 most frequent templates.

Subexpression-isomorphism The effects of subexpression-isomorphism across the corpus are only marginal compared to strict isomorphism. The justifications are reduced to 10,952 templates, which is an overall reduction by 92.2% and a 1% difference compared to strict isomorphism. The number of justifications covered by a single template is slightly increased with an average of 12.9 justifications ($\sigma = 59$, $m = 2$) per template.

The most frequent template (by numbers of justifications) is again Θ_2, which covers 2,128 (1.5% of the total set) justifications in 26 ontologies. Across the ontologies in the corpus, the most frequent template occurs in 28 of the 78 ontologies. This template is a single equivalence axiom which we have already seen



(a) Template frequency for strict iso.

(b) Template frequency for l-iso.

Figure 7.6: Template frequencies for strict and lemma-isomorphism.

in the *Lipid* ontology:

$$\Theta_3 = \{C1 \equiv C2 \sqcap x\} \models C1 \sqsubseteq C2$$

The superfluous part x matches a number of operands such as atomic classes and existential restrictions. Interestingly, while this template occurs in the highest number of ontologies, it only covers 573 justifications across the corpus.

Lemma-isomorphism Across all justifications in the corpus, l-isomorphism has a clearly visible impact. The 141,560 justifications are reduced to only 5,487 templates, which is less than half as many templates as the strictly isomorphic ones, and an overall reduction of 96.9%. Figure 7.6b shows the frequency of templates for lemma-isomorphism.

On average, a template covers 25.8 justifications (σ = 208.5, m =3); however, the large standard deviation shows that the distribution of justifications per template has shifted towards a few very frequent templates, whereas there is still a 'long tail' of 1,878 templates which match only a single justification. If we consider the distribution of justifications per template over the quartiles of the corpus, 25% of the justifications in S_s can be covered by the 8 most frequent templates, 50% by the 44 most frequent templates, and 75% by the 277 (out of 5,487) most frequent templates.

The most frequent templates, by number of justifications they cover, are all subtle variations of a template containing only two or three axioms. Some of these templates, alongside their frequencies (number of justifications the template

Table 7.8: Most frequent templates for lemma-isomorphism across the corpus. #J = number of justifications, #O = number of ontologies.

ID	Template	#J	#O
Θ_4	$\{C1 \sqsubseteq C2, C3 \equiv C2 \sqcup C4, C3 \sqsubseteq C5\} \models C1 \sqsubseteq C5$	7,490	27
Θ_5	$\{C1 \sqsubseteq C2, C5 \equiv C2 \sqcup C3\} \models C1 \sqsubseteq C5$	6,425	28
Θ_6	$\{C1 \sqsubseteq C2, C3 \equiv C2 \sqcup C4 \sqcup C6, C3 \sqsubseteq C5\} \models C1 \sqsubseteq C6$	6,135	26
Θ_7	$\{C1 \sqsubseteq C2, C5 \equiv C2 \sqcup C3 \sqcup C4\} \models C1 \sqsubseteq C5$	4,206	25

covers and number of ontologies the template occurs in), are listed in Table 7.8. Note that any subsumption axiom in the templates Θ_4 through Θ_7 corresponds to an atomic subsumption chain of arbitrary length.

If we look at the number of ontologies a template occurs in, the most frequent templates are Θ_3 and Θ_5, both of which can be found in 28 of the 78 ontologies in the corpus. However, as with strict and subexpression-isomorphism, only a fraction of the templates (8.5%) occur in multiple ontologies, while the majority of templates (5,018 out of 5,487) can only be found in a single ontology.

Isomorphism and superfluity

As we have already seen in our discussion of subexpression-isomorphism, a justification can potentially be non-isomorphic only due to it containing superfluous subexpressions that do not contribute to the entailment.

In order to determine to which extent superfluous parts affect isomorphism, we computed the laconic versions for justifications in S_s, which resulted in a corpus S_{sl} containing 47,667 laconic justifications, of which 31.6% were now atomic subsumption chain justifications or even self-justifications. The number of laconic justifications is significantly lower compared to S_s for two reasons: first, due to the high runtime of the laconic justification generation mechanism, some justifications could not be computed; hence, due to timeouts, the number of entailments in S_{sl} is only 18,300 (compared to 19,097 in S_s). More striking, however, is the effect of removing superfluous expressions on the equality between justifications: multiple regular justifications for individual entailments frequently resulted in *the same* laconic justification, which drastically reduces the number of justifications despite the difference in the number of entailments being only small. Table 7.9 shows a comparison of the cross-corpus reductions for laconic and regular

Table 7.9: Comparison of reductions in S_s and S_{sl}.

	S_s (regular)		S_{sl} (laconic)	
Type	Count	% of S_s	Count	% of S_{sl}
all	141,560	100%	46,667	100%
strict	12,527	8.8%	3,653	7.8%
subex	10,952	7.8%	2,036	6.2%
lemma	5,487	3.1%	1,789	3.8%

justifications in S_s and S_{sl}, respectively.

Across S_{sl}, strict isomorphism reduces the justifications to 3,653 templates, which is a reduction to 7.6% (compared with 8.8% for the regular justifications in S_s). 8 out of the 10 most frequent templates (by number of justifications they cover) are atomic subsumption chains between 1 and 8 axioms. Interestingly, this also indicates that, despite the apparent complexity of many justifications in the corpus (recall that we only computed the laconic versions of *complex* justifications), the actual reasoning behind those justifications comes down to simple atomic subsumption.

Perhaps unsurprisingly, subexpression-isomorphism only has a small effect on S_{sl}. The laconic justifications are reduced to 2,936 templates, which is a reduction to 6.2% of the corpus. Just as with strict isomorphism, this is only a marginally stronger reduction compared to the corpus of regular justifications (7.8%).

Finally, lemma-isomorphism reduces the laconic justifications in S_{sl} to only 1,789 templates, which is an overall reduction to 3.8% of the set of laconic justifications. In terms of the overall proportion this is a smaller reduction than for lemma-isomorphism in regular justifications(3.1%). However, since the set S_{sl} is only roughly a quarter of the size of S_s, this smaller *relative* reduction is not too surprising. As we could already expect based on the prevalent templates we found for strict isomorphism, the most frequent template is a single atomic subsumption axiom, which covers atomic subsumption chain justifications of arbitrary length. This template covers the 15,066 justifications (31.6%) in the corpus we have already mentioned above, and can be found in 61 of the 78 ontologies.

7.3 Discussion

Having presented the major results of our survey of the BioPortal ontologies in the previous section, we will now discuss the significance of the results and the conclusions we can draw from them with respect to the application of structural-based coping strategies for the purpose of ontology debugging.

7.3.1 Justification types and frequency

One of the findings that stands out is the prevalence of atomic subsumption chain justifications: 82% of the entailments in the set of atomic subsumptions and 109 out of 187 ontologies were found to have *only* atomic subsumption chain justifications, while another 12% have atomic subsumption chain justifications in addition to complex justifications. We have seen that ontologies in the weakly expressive OWL 2 EL profile are more likely to contain only trivial entailments. However, DL expressivity alone is not an indicator for complexity, as a number of OWL 2 EL ontologies also produce complex justifications, while some highly expressive ontologies contain no complex justifications.

One explanation for this high number of trivial justifications is the selection of *indirect* atomic subsumptions in the entailment set: as we have shown in 4.1.3, any ontology which is trivial enough to contain only self-justifications for direct atomic subsumptions will only contain atomic subsumption chain justifications for indirect subsumptions. This means that, in general, a large number of justifications a user is likely to encounter will only be trivial self-justifications or atomic subsumption chain justifications.

However we found that approximately one third of the entailments in the corpus have multiple complex justifications, with an average number of nearly 8 justifications per entailment and maximal numbers of up to several hundred (and possibly more). These findings indicate that, while multiple non-trivial justifications are not highly prevalent across the corpus, users have a one in three chance of facing fairly high numbers of non-trivial justifications. As we know from previous research [HBPS11a], even single justifications can be very hard or impossible for users to understand; combined with our insights into the occurrence of multiple justifications, we can conclude that, for the purpose of improving ontology debugging, it is worth focusing on debugging techniques for both individual *and* multiple justifications.

7.3.2 Overlaps

We found a surprisingly high number of high-frequency axioms across the ontologies in the corpus, with all ontologies containing some axiom which occurs in multiple justifications. While the high average frequency of over 50 justifications per axiom is caused by a few high-frequency axioms, the median frequency of 5 justifications shows that shared axioms between justifications are still a frequent occurrence. Since the majority of justifications only has a single entailment, the impact of an axiom is approximately the same as its frequency.

Root and derived justifications are also across the corpus. We have seen that almost three quarters (73.4%) of the justifications in S_s are derived from only a small set of root justifications. The number of entailments that depend on a root justification in S_s is comparatively small (5.4 entailments per root justification), whereas the number of unsatisfiable classes that depend on a root justification in S_u is significantly higher (48.8 entailments). This confirms, in some sense,[9] our existing knowledge of root and derived unsatisfiable classes, which often cites the example of the *Tambis* ontology in which only 3 root unsatisfiable classes were the cause of 144 derived unsatisfiable classes [KPSC06]. On the other hand, the finding also indicates that root and derived justifications have less of an impact on entailments involving *satisfiable* classes.

Our analysis of arbitrary justification overlaps has shown that overlaps with a size and frequency of at least 2 (axioms and justifications, respectively) are frequent occurrences in OWL justifications. The majority (96.1%) of ontologies in the test corpus contained some overlap between complex justifications, with an average overlap size of 5.5 axioms and an average overlap frequency of 7.8 justifications, not taking into account the two ontologies which contribute a large number of high-axiom, high-frequency overlaps. These two outlier ontologies contain axioms with constructs (such as a disjunction with 20 operands in the case of the *Amino Acid* ontology) which pull in many other axioms to form large cores of up to 29 axioms that occur in multiple justifications. The strong connectedness of the justifications in an ontology is confirmed by the small number of graph components, as nearly half of the j-graphs in the test corpus consist of a single connected component, and the average j-graph is made up of only 4 components.

In summary, the frequent occurrence of both single-axiom, root and derived

[9] Taking into account the very small size of S_u and the bias towards one ontology.

relationships, and arbitrary overlap in the corpus indicates that overlap-based debugging techniques, such as highlighting and lemmatising overlaps, will be applicable to a large number of justifications and ontologies. However, we have also seen that the overlaps are mainly restricted to *single* entailments. Considering our goal of using such shared cores for debugging purposes, this means that any overlap-based techniques would be mainly applicable for repairing multiple justifications of single entailments.

7.3.3 Isomorphism

Across all three tests—within entailments, within ontology, and cross-corpus—we have found that strict isomorphism clearly is the most summarising of the isomorphism relations. This shows that a large number of (complex) OWL justifications are structurally *identical*, with many ontologies containing up to 99% structurally identical justifications which can be represented by only a handful of justification templates.

Subexpression-isomorphism, on the other hand, only has little impact on the landscape of justification templates. If we look at the analysis of laconic justifications, only 717 laconic justifications (1.4% of the justifications in S_{sl}) contain constructs (complex expressions that can be substituted by variables) that are affected by subexpression-isomorphism. Note that, since the axioms contain no superfluous parts, any differences in justifications which may have been caused by superfluous expressions no longer hold for this set. This implies that, while subexpression-isomorphism may be helpful in some situations, it is generally not applicable to most justifications. Finally, the finding also indicates that subexpressions are not used propositionally, that is, the semantics of a subexpression in a justification axiom is generally relevant for the entailment (unless the expression is entirely superfluous). If we consider the application of isomorphism-based debugging techniques in OWL tools, the high cost of detecting subexpression-isomorphism (up to two times slower than strict isomorphism) may not be worth paying due to its rather weak effect.

Lemma-isomorphism using atomic subsumption chain lemmatisations, on the other hand, has a more visible effect on justifications. This is particularly significant in our cross-ontology analysis, where lemma-isomorphism reduces the number of templates to only half of those found for strict isomorphism. In particular, we found that 75% of the 141,560 justifications in the corpus can be

covered by only 277 of the most frequent templates, which is a rather stunning result. This clearly shows that justifications which differ in size often only differ in the length of the atomic subsumption chains they contain, but are otherwise strictly isomorphic.

For application in OWL debugging tools, we can conclude that both strict isomorphism and lemma-isomorphism seem promising as the basis for the debugging techniques we have proposed in the previous chapter, while subexpression-isomorphism is only applicable in a small number of ontologies. Furthermore, if we consider only justifications for individual entailments, the average reduction per entailment is significantly weaker than across the entire corpus. This indicates that isomorphism- and template-based techniques may be more suitable for debugging multiple entailments.

7.3.4 Threats to validity

Threats to internal validity Due to the high runtime of the various tasks in the analysis process, it was not possible to obtain complete datasets for all ontologies in practical time. We therefore had to restrict the analysis to random samples and impose timeouts at several stages, which—while statistically significant in terms of numbers—may have caused us to miss some relevant data from ontologies and entailments which were discarded in the process.

First, during the entailment generation process, several ontologies could not be classified in the given time of 20 minutes using the Pellet or JFact reasoners, while for some it was not possible to generate the full set of entailments of the selected type.

In the justification generation stage, the large numbers of entailments for some ontologies, and the performance of the Hitting Set Tree algorithm used in the implementation of the justification generator made it necessary to take a random sample of 1,000 entailments per ontology and generate a maximum of 500 justifications per entailment. The entailment sampling affected 85 of the 187 ontologies that were processed in the justification generation stage.

Further, the overlap analysis was restricted to a sample of 5,000 edges per graph, as for some ontologies the concept lattice generation did not terminate within 24 hours due to the large numbers of connections in the graph.

These sampling strategies affect some of the conclusions we can draw with respect to the justificatory structure of some of the ontologies in the corpus. Take,

for example, the *Bone Dysplasia* ontology, a large and expressive (44,683 axioms) $\mathcal{SHIF}$ ontology in the corpus, for which we generated over 100,000 entailments. Due to the sampling process, the analysis covered only 800 entailments whose justifications contained 882 axioms. It is clear to see that we cannot make any conclusive statements about certain metrics, such as the activity of this ontology, based on the small sample obtained since the activity is based on the total number of justification axioms.

Finally, as we have dicussed in Chapter 5, we have not yet shown that lemma-isomorphism (restricted to maximal atomic subsumption chains) is in fact transitive. This means that we might potentially over-estimate the logical diversity of a corpus and under-estimate the effects of lemma-isomorphism. However, we can simply treat the results as a lower bound to the reductions caused by l-isomorphism without running risk of reporting results that are 'too good'.

Threats to external validity While the BioPortal corpus does contain a number of *interesting* (in terms of size and complexity) ontologies, it may not be representative for most OWL ontologies used in practice. Due to their intended application in research and their being made available in a curated portal it is certainly possible that these ontologies are potentially larger and crafted more 'carefully' (i.e. use more complex constructors) than most OWL ontologies found on the web. Indeed, a random sample of ontologies obtained from a web crawl [GMPS13] was found to contain a significantly smaller number of axioms ($m = 57$) than the BioPortal ontologies.

Overall, we can say that the statistics obtained from the BioPortal corpus allow us to draw conclusions regarding general trends in justificatory structure in a set of OWL ontologies that were (mainly) authored manually. While certain statistics regarding correspondence between justificatory structure and general ontology metrics are to be taken with care, the majority of our findings are based on large enough random samples to be statistically significant with respect to the general occurrence of justifications.

7.4 Summary and conclusions

In this chapter, we have presented a survey of the landscape of justificatory structure in OWL ontologies. Using an initial set of over 300 ontologies from the

NCBO BioPortal which was reduced down to 78 ontologies containing complex justifications, we analysed the frequency and extent of various features of justificatory structure. We have found surprisingly high degrees of overlap between the justifications for *single* entailments, with an average overlap frequency of 11 justifications and an average size of 12 justifications per overlap. Likewise, we have seen that root and derived relationships occur frequently across the corpus, but mainly affect justifications for *individual* entailments.

Strict justification isomorphism was shown to be prevalent throughout the corpus, as a large number of strictly isomorphic justifications cause a reduction of the justification corpus to less than 10% of its original size. In contrast, subexpression-isomorphism has the least effect on the justifications, with only marginal differences between strict and subexpression-isomorphism, whereas lemma-isomorphism captures isomorphic justifications in the majority of ontologies in the corpus.

These results are highly significant, as they demonstrate that OWL ontologies used in practice can—and do— indeed have a very rich justificatory structure, with frequent overlaps at surprisingly large degrees, high-frequency axioms, and structural similarity. On the other hand, we have also seen that some of the proposed interventions may not be applicable in some cases; for example, the number of justifications that have multiple entailments is fairly low and affects only a fraction of the ontologies in the corpus, while overlap between justifications occurs mainly for single, but not multiple entailments.

Chapter 8

Conclusions

In this final chapter, we will summarise the work and results presented in this thesis. We will discuss the main contribution of this thesis and the significance of the results and insights gained, while also highlighting some open issues. Finally, we will give an outlook on future directions for further research into the logical and cognitive aspects related to justifications for entailments of OWL ontologies.

8.1 Summary of contributions

In summary, this thesis introduced the notion of justificatory structure, provided definitions for the different aspects of structure, proposed strategies for exploiting the justificatory structure of an ontology in order to provide improved debugging support, and analysed a set of ontologies from the bio-medical domain to determine the prevalence and extents of structural phenomena in OWL ontologies used in practice.

8.1.1 Design decisions for finite entailment sets

We first discussed the issue of limiting the infinite set of entailments of an ontology to a finite entailment set and proposed several design decisions to be made in order to arrive at a sensible *representation* of a finite entailment set. These design decisions were largely motivated by the issue of *counting* entailments, e.g. for the purpose of comparing the inferential power of two ontologies, which requires the number of entailments to grow monotonically when adding axioms to the ontology.

We found that certain design decisions cause the number of entailments to grow non-monotonically: not including asserted axioms or indirect subsumptions in a finite entailment set, for instance, can lead to *fewer* entailments in the set if we add axioms to an ontology. We also provided a practical definition for distinguishing imported, mixed, and native entailments based on their justifications.

Finally, we provided a shorthand notation for referring to a specific representation of a finite entailment set and demonstrated the effect of different design decisions toy examples and real-life applications of finite entailment sets of OWL ontologies.

8.1.2 Justificatory structure and justification isomorphism

In Chapters 4 and 5, we introduced the notion of justificatory structure using a graph representation of the justifications, justification axioms, and entailments for a given set of entailments and justifications in an ontology. Structural aspects include graph metrics such as the in- and out-degrees of justification and axiom nodes, graph components, and overlap of varying degrees. Overlap between justifications, that is, shared axiom sets, are of particular interest in the context of ontology debugging, as they potentially lead to smaller repairs, while also helping the understanding of multiple justifications through lemmas.

We then moved on to discuss the issue of structural similarities between justifications and extended the existing notion of justification isomorphism to two new equivalence relations, subexpression-isomorphism $\approx_s$ and lemma-isomorphism $\approx_\ell$. Under (strict) isomorphism, we consider two justifications to be equivalent if they use the same constructors and axiom types and only differ in the class, property, and individual names they use. Subexpression-isomorphism extends this notion to cover justifications which differ in the expressions they use if those expressions can be substituted by freshly generated variable names; that is, if their complex subexpressions are used in a *propositional* way. Lemma-isomorphism applies the principle of subexpression-isomorphism to a lemmatised justifications, that is, a justification in which a subset S was replaced with an entailment λ of S.

The two new notions of isomorphism were designed to be 'upwards-compatible', that is, (strictly) isomorphic justifications are subexpression-isomorphic, and subexpression-isomorphic justifications are considered to be lemma-isomorphic. We showed that subexpression-isomorphism is indeed an equivalence relation under certain (non-obvious) side conditions, and provided a proof for the transitivity of subexpression-isomorphism.

8.1.3 Reducing user effort

In Chapter 6, we looked at the problem of debugging erroneous entailments in OWL ontologies, providing a definition for 'debugging problems' and how to determine whether a modification successfully solved the debugging problem. We constructed a simple model for measuring the *effort* involved in solving a debugging problem using justification-based explanation tools. This model also allows us to measure whether a debugging strategy *reduces* the effort involved in debugging a set of entailments by introducing an *alleviation factor a*.

We proposed several strategies for exploiting the justificatory structure of an ontology in order to reduce the user effort when faced with multiple justifications. These strategies include presenting the user with high-ferquency axioms, enriching justifications with common lemmas originating from overlaps, and presenting the user with an abstract justification template of isomorphic justifications.

8.1.4 Experimental results

Finally, we presented a survey of OWL and OBO ontologies from the NCBO BioPortal. We found that the majority of justifications for direct and indirect atomic susbumptions between satisfiable classes can be classified as 'atomic subsumption chain' justifications, whereas the number of 'complex' justifications found across the corpus is comparatively low. For those entailments that *do* have complex justifications, the number of justifications per entailment was found to be surprisingly high, with an average of 8 justifications per entailment and 70% of the entailments in the corpus having *multiple* complex justifications. Shared axioms and axiom sets were found to occur frequently across the corpus: Over 73% of the justifications in the corpus were derived from some other justification (which may be partially due to the inclusion of indirect subsumptions in the entailment set). Arbitrary overlap between justifications for *single* entailments is highly prevalent in the corpus, with the majority of overlaps having a size of around 5 axioms and occurring in around 5 justifications.

Our analysis of justification isomorphism in the BioPortal corpus revealed that justification isomorphism between justifications for *individual* entailments occurs fairly frequently: Strict isomorphism applied to justifications for individual entailments causes an average reduction from justifications to templates by

33%, whereas subexpression-isomorphism has only marginal effects in some ontologies. However, if we consider the logical diversity of all justifications for all entailments in the corpus, strict isomorphism and lemma-isomorphism have a significant impact: the 141,560 justifications in the corpus are effectively reduced to just over 12,500 templates (a reduction of 91.2%) by strict isomorphism, and lemma-isomorphism finally reduces this number to only 5,487 templates. More strikingly, we have found that 75% of the justifications in the corpus can be covered by only 277 of the most frequent templates for lemma-isomorphism. Across the set of laconic versions of the justifications in the corpus, the *relative* effects of the three isomorphism types are roughly the same; however, the final number of templates for lemma-isomorphism is considerably lower at only 1,789 templates. This shows that the logical diversity of justifications is far lower than their material manifestiations, as removing all superfluous parts in a corpus of 141,560 regular (complex) justifications reduces it to just over 1% of its original size.

8.2 Significance of results

Against the backdrop of justification-based debugging support, this thesis has advanced the state of knowledge we have of the relations between justifications in OWL ontologies. It highlighted possible new strategies for generating OWL ontology metrics and for providing improved debugging support, which lays the foundations for future approaches to building user-friendly and efficient OWL ontology tools for both ontology analysis and ontology development.

First, the design decisions we provided for generating and representing finite entailment sets draw from the multitude of modifications and 'hacks', as well as misunderstandings we have encountered in OWL tools and analytical applications. Previously, there has been no clear account of the various factors that have an impact on the size of entailment sets and no convenient way of referring to a certain type of entailment set, which we believe to have rectified with the design decisions and shorthand notation provided in this thesis.

This has been the first in-depth investigation of the justificatory structure of OWL ontologies. In our survey of the BioPortal ontologies, we have found that, while a large number of entailments only have trivial atomic subsumption chain justifications, a significant number of entailments indeed has multiple complex justifications. This finding shows that improved debugging support for dealing

with multiple justifications is clearly necessary, as users have a high chance of encountering an entailment that has multiple non-trivial justifications.

While root and derived justifications and axiom frequency have been (somewhat implicitly) used in justification-based debugging tools, there has been no extension of these relations to cover arbitrary overlap. There have been some previous experiments regarding root and derived relationships between justifications [KPSH05, MMV10], however, the survey presented in this thesis is the first large-scale experiment investigating the occurrence of both root and derived justifications and arbitrary justification overlap in OWL ontologies. We have found that all types of overlap occur frequently, which provides us with important knowledge of the structural relations between justifications and informs future ontology debugging tools, which can make use of these structural aspects.

Thus far, justification isomorphism has only been mentioned in the context of sampling justifications for a user study [HBPS11a]. The results presented in our survey of justification isomorphism in the BioPortal corpus confirms that a) isomorphic justifications can be determined in practical time even using a naive implementation, and b) a large number of justifications are structurally isomorphic. This shows that template-based debugging support is both feasible for and widely applicable to justifications found in practice.

However, we have also found that the newly introduced relations, subexpression-isomorphism and lemma-isomorphism, do not have as big an impact on the BioPortal justifications as we would have hoped. S-isomorphism in particular only affects a fraction of the justifications in the corpus, whereas lemma-isomorphism occurs in most ontologies, but only in small numbers. This implies that, for most justifications, strict isomorphism may already be sufficient for finding a common template. On the other hand, the small effect of s-isomorphism also shows us that there exists *real* logical diversity in the modelling of ontologies, and that complex subexpressions are generally *not* used in a propositional way. Furthermore, while we could successfully prove the transitivity of s-isomorphism, the transitivity of l-isomorphism and the selection of suitable lemmatisations remains an open question. In order to make l-isomorphism useful in OWL applications, we need to further explore potential lemmatisations which are guaranteed to preserve the transitivity of l-isomorphism.

With the expection of root and derived justifications and a focus on minimal repairs, most of the existing justification-based debugging techniques do not

consider the issue of multiple justifications for repair and treat justifications as isolated entities. By suggesting structure-based coping strategies for multiple justifications we have made a step towards improved debugging support which is targeted at multiple justifications. The introduction of a model for measuring *and quantifying* the success of a coping strategy in particular paves the way for principled empirical research into the effects of different justification-based debugging techniques.

8.3 Future directions

While we have covered a broad range of topics in this thesis, it is clear that there is plenty of room for future work. In this section we will outline potential extensions of this work, which cover the theoretical foundations of justificatory structure, further experiments, as well as applications of the strategies proposed in this work.

Lemma-isomorphism We have made some progress towards defining lemma-isomorphism and finding suitable lemmatisations; however, there are still some open questions remaining. First of all, we have restricted the lemmatisations to (maximal) atomic subsumption chains in order to demonstrate the concept of l-isomorphism, which has already had some visible effects. On the other hand, if we think back to the original motiviation for l-isomorphism—the *Pizza* ontology in which there exists several similar reasons for why some Pizza is a subclass of InterestingPizza—we can see that atomic subsumption chain lemmatisations do not cover the justifications we can find there. Thus, extending the set of lemmatisations to be used in l-isomorphism based on their *obviousness* for OWL users seems to be an important next step, in particular since we have shown that over three quarters of the justifications in the test corpus could be covered by a strikingly small number of templates for lemma-isomorphism. Adding only a small number of additional lemmatisations may already be sufficient to cover the vast majority of justification shapes found in OWL ontologies.

However, since we have seen that preserving the transitivity of our isomorphisms is rather non-trivial, this will also require a thorough investigation of the conditions lemmatisations have to meet in order for l-isomorphism to be transitive.

Dealing with masking and superfluity While we discussed issues such as non-laconic justifications and justification masking where applicable, our introduction to justificatory structure and the survey of the BioPortal ontologies did not fully explore the effects of masking and superfluity. We know that masking and superfluity are frequent occurrences in OWL ontologies [Hor11a] and that superfluous parts may cause users difficulties in understanding justifications [HBPS11a]. In our analysis of isomorphism we also found that a large number of justifications are considered to be non-isomorphic due to them containg superfluous parts, and that laconicising justifications significantly reduces the overall diversity of justifications. A more in-depth comparison between the justificatory structure of non-laconic and laconic justification sets will help us gain a better understanding of how these phenomena affect the justificatory structure of an ontology, and to which extent this has an impact on the debugging process.

Effects on justification computation Another open question that arises from the work presented in this thesis is the effect of justificatory structure on justification computation, and, in a wider sense, reasoning performance. While we took a brief glance at the effects of graph components on the performance of justification computation, we omitted an in-depth investigation of the relations between structure, hitting set tree size, calls to a 'find one' subroutine, and computation times. This will require further experiments and artificial generation of justifications to test the effects of various degrees of justification overlap, activity, and graph structure in isolation. Further insights into the relation between justificatory structure and justification computation would potentially allow us to generate guidelines, ontology design patterns, or even automated tool features to yield easily computable justifications. Such guidelines might either be in line with existing ontology design patterns, or potentially require users to sacrifice some aspects of 'good modelling' for the sake of easy justification computation; thus, an investigation of the interplay between existing ontology design patterns and justificatory structure is another aspect of justificatory structure worth exploring.

Further, since justifications are, in some sense, responsible for the subsumption relationships in an ontology and therefore its class hierarchy, we may also ask: how does the justificatory structure of an ontology affect the classification performance of an OWL reasoner? We have made some steps into this direction with JustBench [BPS10a], a justification-based micro-benchmarking tool for

OWL reasoners; however, in JustBench we only used justification-entailment pairs in order to measure the performance of entailment checking on naturally arising ontology subsets, which did not involve any analysis of justificatory structure.

Visualisation and user interaction Possibly one of the most intriguing directions for future work is the exploration of user interaction mechanisms to exploit the strategies presented in this thesis for implementation in OWL tools. We have made a first attempt at outlining interaction mechanisms for multiple justifications in Chapter 6. However, the design and implementation of a *useful* OWL debugging tool will require significantly more research into the cognitive aspects of how users read and understand OWL constructs and axioms, starting with the fundamental question of whether OWL users build *mental models* [JL80] of the information they digest, or whether they apply *rules* and symbol manipulation in order to understand reasoning processes, and to which extent this behaviour depends on the user *profile* and their specific *task*. While we have gained some insights in recent years into how people process OWL and what causes them difficulties in understanding, we are far from having a clear picture of the cognitive processes associated with interacting with OWL ontologies.

Once we have a better understanding of these cognitive processes as well as the demands and requirements of OWL users with different backgrounds, we need to identify a suitable approach to visualising and interacting with OWL ontologies and justifications. There have been various visualisation tools for (particular aspects of) OWL ontologies, such as OWLViz[1] and CropCircles [WP06] which visualise the class hierarchy of an OWL ontology, SuperModel [BSP09] which displays segments of a model of the ontology, and DeMost [DKP+11], which represents the atomic decomposition of an ontology as a graph of axioms and axiom sets.

However, none of these approaches (with the exception of CropCircles perhaps which uses a non-standard rendering of nested circles to represent subsumption relationships) seem to have grown out of a focussed investigation of the suitability of different interaction mechanisms for the task at hand; we may argue that graph-based approaches were chosen because they are simply the most straightforward way of representing the types of relations occurring in an OWL ontology.

[1] `http://www.co-ode.org/downloads/owlviz/`

Given the progress made in interactive data visualisation tools, such investigations would be both highly interesting and worthwhile, as improved tools for OWL ontology building and debugging may as well contribute to the wider acceptance and propagation of OWL.

Applications of justifications Finally, another topic we have only touched upon in this thesis is the application of justifications beyond debugging and repair. One of these areas is ontology comprehension and learning, that is, OWL users wanting to familiarise themselves with an unknown ontology, or OWL novices attempting to understand the implications of reasoning in OWL ontologies. We have seen that many aspects of justificatory structure, such as justification overlap and isomorphism, reveal deeper insights of the interactions between axioms in an ontology, while also providing information akin to ontology *patterns*. We can imagine using aspects of justificatory structure to teach and learn 'reasoning patterns' in OWL ontologies, which may support OWL novices in building ontologies.

Further, the justificatory structure of an OWL ontology, such as the number of justifications, the number and sizes of overlaps, and the structural similarities between justifications, provides us with *implicit* ontology metrics. The number of justifications per entailment has already been used as an ontology metric to determine the *justification richness* of an ontology [MSR11], and we have shown how two seemingly similar ontologies with approximately the same number of classes and axioms can differ vastly in their justificatory structure [BHPS11]. Integrating justification-based metrics into OWL tools will make these differences visible to OWL developers, allowing them to compare and rank ontologies based on an extensive set of both explicit and implict metrics.

Bibliography

[AAKS08] Mikel Egaña Aranguren, Erick Antezana, Martin Kuiper, and Robert Stevens. Ontology design patterns for bio-ontologies: a case study on the cell cycle ontology. *BMC Bioinformatics*, 9(S-5), 2008. (Cited on page 37.)

[AB06] Harithand Alani and Christopher Brewster. Metrics for ranking ontologies. In *Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON-06)*, pages 24–30, 2006. (Cited on page 19.)

[ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009. (Cited on page 32.)

[AGU72] Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. (Cited on page 64.)

[All05] Dean Allemang. S is for semantics. `http://dallemang.typepad.com/my_weblog/2005/06/ive_been_progra.html` [Accessed: 22-03-2013], 2005. (Cited on page 132.)

[BB07] Martins Barinskis and Guntis Barzdins. Satisfiability model visualization plugin for deep consistency checking of OWL ontologies. In *Proceedings of the 3rd International Workshop on OWL: Experiences and Directions (OWLED-07)*, 2007. (Cited on page 57.)

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 364–369, 2005. (Cited on pages 25 and 32.)

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic*

Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003. (Cited on pages 11 and 22.)

[BFH99] Alex Borgida, Eric Franconi, and Ian Horrocks. Explaining $\mathcal{ALC}$ subsumption. In *Proceedings of the 12th International Workshop on Description Logics (DL-99)*, 1999. (Cited on pages 13, 39, and 54.)

[BG94] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *The Journal of Logic Programming*, 19–20, Supplement 1:73–148, 1994. (Cited on page 22.)

[BGSS07] Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 230–235, 2007. (Cited on page 36.)

[BH95] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14(1):149–180, 1995. (Cited on page 43.)

[BHPS11] Samantha Bail, Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The justificatory structure of the NCBO bioportal ontologies. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*, 2011. (Cited on pages 15, 21, 33, 53, 80, 82, 142, 143, 145, and 180.)

[Bie07] Meghyn Bienvenu. Consequence finding in $\mathcal{ALC}$. In *Proceedings of the 20th International Workshop on Description Logics (DL-07)*, 2007. (Cited on pages 69 and 70.)

[Bie09] Meghyn Bienvenu. Prime implicates and prime implicants: From propositional to modal logic. *Journal of Artificial Intelligence Research*, 36:71–128, 2009. (Cited on page 69.)

[BKBM99] Franz Baader, Ralf Küsters, Alex Borgida, and Deborah L. McGuinness. Matching in description logics. *Journal of Logic and Computation*, 9(3):411–447, 1999. (Cited on page 103.)

[BKM99] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 96–103, 1999. (Cited on page 139.)

[BL04] Ronald J. Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2004. (Cited on page 22.)

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001. (Cited on page 32.)

[BM09] Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA-09)*, pages 350–364, 2009. (Cited on page 103.)

[BN98] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 331–335, 1998. (Cited on page 103.)

[BP08a] Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR-08)*, pages 226–241, 2008. (Cited on page 43.)

[BP08b] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 2008. (Cited on page 41.)

[BP10] Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. *Journal of Automated Reasoning*, 2010. (Cited on page 43.)

[BPS07a] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}+$. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 52–67, 2007. (Cited on pages 40, 43, and 53.)

[BPS07b] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI-07)*, pages 52–67, 2007. (Cited on pages 41 and 42.)

[BPS10a] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Justbench: A framework for OWL benchmarking. In *Proceedings of the 9th International Semantic Web Conference (ISWC-10)*, 2010. (Cited on page 178.)

[BPS10b] Samantha Bail, Bijan Parsia, and Ulrike Sattler. The justificatory structure of OWL ontologies. In *Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED-10)*, 2010. (Cited on pages 21 and 82.)

[BPS11] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Extracting finite sets of entailments from OWL ontologies. In *Proceedings of the 24th International Workshop on Description Logics (DL-11)*, 2011. (Cited on page 21.)

[BPS12a] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Declutter your justifications: Determining similarity between owl explanations. In *Proceedings of the First International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM-12)*, 2012. (Cited on page 21.)

[BPS12b] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Diversity of reason: Equivalence relations over description logic explanations. In *Proceedings of the 25th International Workshop on Description Logics (DL-12)*, 2012. (Cited on page 21.)

[Bra04] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pages 298–302, 2004. (Cited on page 25.)

[BS84a] Bruce G. Buchanan and Edward H. Shortliffe. Explanation as a topic of AI research. In Bruce G. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems*. Addison-Wesley, 1984. (Cited on page 12.)

[BS84b] Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule-Based Expert Systems*. Addison-Wesley, 1984. (Cited on pages 12 and 13.)

[BS00] Franz Baader and Ulrike Sattler. Tableau algorithms for description logics. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-00)*, pages 1–18, 2000. (Cited on page 27.)

[BSP09] Johannes Bauer, Ulrike Sattler, and Bijan Parsia. Explaining by example: Model exploration for ontology comprehension. In *Proceedings of the 22nd International Workshop on Description Logics (DL-09)*, 2009. (Cited on pages 18, 57, 124, and 179.)

[BST07] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *Journal of Applied Logic*, 5(3):392 – 420, 2007. (Cited on page 139.)

[Car07] Jorge Cardoso. The semantic web vision: Where are we? *IEEE Intelligent Systems*, 22(5):84–88, 2007. (Cited on page 32.)

[CGL+11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semantic Web Journal*, 2(1):43–53, 2011. (Cited on page 32.)

[CHKS07] Bernardo Cuenca Grau, Ian Horrocks, Yvgeny Kazakov, and Ulrike Sattler. Just the right amount: Extracting modules from ontologies. In *Proceedings of the 14th International World Wide Web Conference (WWW-05)*, 2007. (Cited on page 44.)

[CHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, 31, 2008. (Cited on page 44.)

[CHM+08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 2008. (Cited on pages 11 and 22.)

[Cla81] William J. Clancey. The epistemology of a rule-based expert system: a framework for explanation. Technical report, Stanford University, Stanford, CA, USA, 1981. (Cited on page 12.)

[CPSK06] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Modularity and web ontologies. In *Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning (KR-06)*, 2006. (Cited on page 44.)

[CRVBP09] Oscar Corcho, Catherine Roussey, Luis Manuel Vilches-Blázquez, and Iván Pérez. Pattern-based OWL ontology debugging guidelines. In *Proceedings of the Workshop on Ontology Patterns (WOP 2009)*, 2009. (Cited on page 129.)

[Dav81] Martin Davis. Obvious logical inferences. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 530–531, 1981. (Cited on pages 55 and 112.)

[dCHS+04] Sherri de Coronado, Margaret W. Haber, Nicholas Sioutos, Mark S. Tuttle, and Lawrence W. Wright. NCI Thesaurus: Using science-based terminology to integrate cancer research results. *Studies in Health Technology and Informatics*, 107(1), 2004. (Cited on page 11.)

[DHS05] Xi Deng, Volker Haarslev, and Nematollaah Shiri. A framework for explaining reasoning in description logics. In *Working Notes of the AAAI Fall Symposium on Explanation-Aware Computing*, 2005. (Cited on page 39.)

[DKP+11] Chiara Del Vescovo, Pavel Klinov, Bijan Parsia, Ulrike Sattler, and Thomas Schneider. DeMoSt: a tool for exploring the decomposition and the modular structure of OWL ontologies. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*, 2011. (Cited on page 179.)

[DL96] Giuseppe De Giacomo and Maurizio Lenzerini. Tbox and Abox reasoning in expressive description logics. In *Proceedings of the 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 316–327, 1996. (Cited on page 27.)

[DQJ09] Jianfeng Du, Guilin Qi, and Qiu Ji. Goal-directed module extraction for explaining OWL DL entailments. In *Proceedings of the 8th International Semantic Web Conference (ISWC-09)*, pages 163–179, 2009. (Cited on pages 41 and 44.)

[DS08] Jianfeng Du and Yi-Dong Shen. Computing minimum cost diagnoses to repair populated DL-based ontologies. In *Proceedings of the 17th International World Wide Web Conference (WWW-08)*, 2008. (Cited on page 56.)

[FH88] Amy Felty and Greg Hager. Explaining modal logic proofs. In *Proceedings of the IEEE 1988 International Conference on Systems, Man, and Cybernetics*, 1988. (Cited on page 55.)

[Fit83] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Springer, 1983. (Cited on page 55.)

[FMV10] Enrico Franconi, Thomas Meyer, and Ivan Varzinczak. Semantic diff as the basis for knowledge base versioning. In *Proceedings of the 13th International Workshop on Non-Monotonic Reasoning (NMR-2010)*, 2010. (Cited on page 36.)

[FS05] Gerhard Friedrich and Kostyantyn Shchekotykhin. A general diagnosis method for ontologies. In *Proceedings of the 4th International Semantic Web Conference (ISWC-05)*, pages 232–246, 2005. (Cited on pages 43 and 56.)

[Gan05] Aldo Gangemi. Ontology design patterns for semantic web content. In *Proceedings of the 4th International Semantic Web Conference (ISWC-05)*, pages 262–276, 2005. (Cited on page 37.)

[Gär88] Peter Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, 1988. (Cited on page 40.)

[GLC+01] Fernand Gobet, Peter C.R. Lane, Steve Croker, Peter C-H. Cheng, Gary Jones, Iain Oliver, and Julian M. Pine. Chunking mechanisms in human learning. *Trends in Cognitive Sciences*, 5(6):236–243, 2001. (Cited on page 137.)

[GMPS13] Rafael S. Gonçalves, Nicolas Matentzoglu, Bijan Parsia, and Uli Sattler. The empirical robustness of description logic classification. In *Submitted to the 26th International Workshop on Description Logics*, 2013. (Cited on page 170.)

[GPS11a] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Analysing the evolution of the NCI thesaurus. In *Proceedings of the 24th IEEE International Symposium on Computer-Based Medical Systems (CBMS-11)*, 2011. (Cited on page 37.)

[GPS11b] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Categorising logical differences between OWL ontologies. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM-11)*, 2011. (Cited on pages 36 and 79.)

[GPS12a] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Concept-based semantic difference in expressive description logics. In *Proceedings of the 25th International Workshop on Description Logics (DL-12)*, 2012. (Cited on page 79.)

[GPS12b] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Performance heterogeneity and approximate reasoning in description logic ontologies. In *Proceedings of the 11th International Semantic Web Conference (ISWC-12)*, 2012. (Cited on page 38.)

[Gru93] Thomas Gruber. Towards principles for the design of ontologies used for knowledge sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation*, 1993. (Cited on page 30.)

[GSG04] Pierre Grenon, Barry Smith, and Louis Goldberg. Biodynamic ontology: applying the BFO in the biomedical domain. In *Ontologies in Medicine*. IOS Press, 2004. (Cited on page 80.)

[GSW89] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989. (Cited on page 43.)

[GSW05] Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal Concept Analysis: Foundations and Applications*. Springer, 2005. (Cited on pages 36 and 154.)

[GW02] Nicola Guarino and Christopher A. Welty. Evaluating ontological decisions with OntoClean. *Commununications of the ACM*, 45(2):61–65, 2002. (Cited on page 56.)

[GW04] Nicola Guarino and Christopher A. Welty. An overview of OntoClean. In *Handbook on Ontologies*, pages 151–172. Springer, 2004. (Cited on page 56.)

[GWS07] Andrew Gibson, Katy Wolstencroft, and Robert Stevens. Promotion of ontological comprehension: Exposing terms and metadata with web 2.0. In *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007)*, 2007. (Cited on page 124.)

[HBMB05] Graeme S. Halford, Rosemary Baker, Julie E. McCredden, and John D. Bain. How many variables can humans process? *Psychological Science*, 16(1):70–76, 2005. (Cited on page 137.)

[HBPS11a] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*, 2011. (Cited on pages 15, 21, 52, 95, 100, 102, 112, 124, 131, 142, 166, 176, and 178.)

[HBPS11b] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In *Proceedings of the 24th International Workshop on Description Logics (DL-11)*, 2011. (Cited on pages 21, 52, and 129.)

[HBPS13] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. Toward cognitive support for OWL justifications. *Knowledge-Based Systems (in submission)*, 2013. (Cited on page 21.)

[HDG+06] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The Manchester OWL syntax. In *Proceedings of the 2nd International Workshop on OWL: Experiences and Directions (OWLED-06)*, 2006. (Cited on page 31.)

[Hen07] James Hendler. The dark side of the semantic web. *IEEE Intelligent Systems*, 22(1):2–3, 2007. (Cited on page 33.)

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning (KR-06)*, 2006. (Cited on pages 11, 25, and 30.)

[Hor97] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997. (Cited on page 27.)

[Hor02] Ian Horrocks. DAML+OIL: a description logic for the semantic web. *IEEE Data Engineering Bulletin*, 25(1):4–9, 2002. (Cited on pages 30 and 32.)

[Hor10] Matthew Horridge. OWL syntaxes. `http://ontogenesis.knowledgeblog.org/88` [Accessed: 2012-06-19], 2010. (Cited on page 31.)

[Hor11a] Matthew Horridge. *Justification Based Explanation in Ontologies*. PhD thesis, The University of Manchester, 2011. (Cited on pages 44, 45, 48, 50, 82, 89, 110, 115, 130, and 178.)

[Hor11b] Matthew Horridge. A practical guide to building OWL ontologies using Protégé 4 and CO-ODE tools, edition 1.3. `http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/` [Accessed: 30-09-2012], 2011. (Cited on page 34.)

[HP09] Matthew Horridge and Bijan Parsia. From justifications to proofs for entailments in OWL. In *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED-09)*, 2009. (Cited on page 51.)

[HPS08] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. In *Proceedings of the 7th International Semantic Web Conference (ISWC-08)*, pages 323–338, 2008. (Cited on pages 13, 15, 31, 48, 115, and 140.)

[HPS09] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in OWL. In *Proceedings of the 22nd International Workshop on Description Logics (DL-09)*, 2009. (Cited on pages 34, 82, 129, and 131.)

[HPS10a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification masking in OWL. In *Proceedings of the 23rd International Workshop on Description Logics (DL-10)*, 2010. (Cited on pages 48, 49, and 115.)

[HPS10b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification oriented proofs in OWL. In *Proceedings of the 9th International Semantic Web Conference (ISWC-10)*, pages 354–369, 2010. (Cited on pages 15 and 51.)

[HPS11] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The state of biomedical ontologies. In *Proceedings of the 2011 ISMB Bio-Ontologies SIG*, 2011. (Cited on pages 12 and 33.)

[HPS12] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Extracting justifications from BioPortal ontologies. In *Proceedings of the 11th International Semantic Web Conference (ISWC-12)*, 2012. (Cited on page 33.)

[HPSv03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. (Cited on pages 22 and 25.)

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR-99)*, pages 161–180, 1999. (Cited on page 25.)

[HST00] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000. (Cited on page 27.)

[HvT05] Zhisheng Huang, Frank van Harmelen, and Annette Teije. Reasoning with inconsistent ontologies. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005. (Cited on page 30.)

[Jac92] Peter Jackson. Computing prime implicates. In *Proceedings of the 1992 ACM Annual Conference on Communications*, pages 65–72, 1992. (Cited on page 69.)

[JGSV06] Cliff Joslyn, Damian Gessler, Stefan Schmidt, and Karin Verspoor. Distributed representations of bio-ontologies for semantic web services. In *Joint BioLink and BioOntologies SIG of ISMB 2006*, 2006. (Cited on page 64.)

[JL80] Philip N. Johnson-Laird. Mental models in cognitive science. *Cognitive Science*, 4(1):71–115, 1980. (Cited on pages 130 and 179.)

[JQH08] Qiu Ji, Guilin Qi, and Peter Haase. A relevance-based algorithm for finding justifications of DL entailments. Technical report, University of Karlsruhe, 2008. (Cited on pages 41 and 53.)

[Kal06] Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland at College Park, 2006. (Cited on pages 43, 93, 131, 133, and 135.)

[KDLD08] Mykola Konyk, Alexander De Leon, and Michel Dumontier. Chemical knowledge for the semantic web. In *Data Integration in the Life Sciences*, pages 169–176. Springer Berlin / Heidelberg, 2008. (Cited on page 80.)

[Kee07] C. Maria Keet. Enhancing comprehension of ontologies and conceptual models through abstractions. *Proceedings of the 10th Conference of the Italian Association for Artificial Intelligence (AI*IA-07)*, pages 813–821, 2007. (Cited on page 18.)

[KKS11] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. Unchain my $\mathcal{EL}$ reasoner. In *Proceedings of the 24th International Workshop on Description Logics (DL-11)*, 2011. (Cited on page 31.)

[KPC06] Aditya Kalyanpur, Bijan Parsia, and Bernardo Cuenca Grau. Beyond asserted axioms: Fine-grain justifications for OWL-DL entailments. In *Proceedings of the 19th International Workshop on Description Logics (DL-06)*, 2006. (Cited on pages 15 and 47.)

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *Proceedings of the 6th International Semantic Web Conference (ISWC/ASWC-07)*, pages 267–280, 2007. (Cited on pages 43, 44, and 142.)

[KPS+06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics*, 4(2):144–153, 2006. (Cited on page 47.)

[KPSC06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in OWL ontologies. In *Proceedings of the 3rd European Semantic Web Conference (ESWC-06)*, pages 170–184, 2006. (Cited on pages 36, 46, and 167.)

[KPSH05] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005. (Cited on pages 13, 16, 40, 42, 43, 52, 53, 54, 131, and 176.)

[KŠK11] Petr Křemen, Marek Šmíd, and Zdenek Kouba. OWLDiff: A practical tool for comparison and merge of OWL ontologies. In *Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA-11)*, 2011. (Cited on page 36.)

[Kwo05] Francis Kwong. Practical approach to explaining $\mathcal{ALC}$ subsumption. Master's thesis, The University of Manchester, 2005. (Cited on page 43.)

[Lam07] Joey Sik Chun Lam. *Methods for Resolving Unsatisfiable Ontologies*. PhD thesis, University of Aberdeen, 2007. (Cited on pages 13 and 52.)

[LH05] Thorsten Liebig and Michael Halfmann. A tableau-based explainer for DL subsumption. In *Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-05)*, pages 323–327, 2005. (Cited on page 43.)

[Lin89] Christoph Lingenfelder. Structuring computer generated proofs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 378–383, 1989. (Cited on page 55.)

[LMS04] Inês Lynce and João Marques-Silva. On computing minimum unsatisfiable cores. In *International Symposium on Theory and Applications of Satisfiability Testing*, pages 305–310, 2004. (Cited on page 40.)

[LN04] Thorsten Liebig and Olaf Noppens. OntoTrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. In *Proceedings of the 3rd International Semantic Web Conference (ISWC-04)*, pages 244–257, 2004. (Cited on page 39.)

[LPSV06] Joey Sik Chun Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. In *Proceedings of the 5th IEEE/WIC/ACM International Conference on Web Intelligence (WI-06)*. Springer, 2006. (Cited on pages 15 and 47.)

[LS08] Harris Lin and Evren Sirin. Pellint - a performance lint tool for Pellet. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED-08EU)*, 2008. (Cited on page 38.)

[Lut02] Carsten Lutz. *The Complexity of Description Logics with Concrete Domains*. PhD thesis, RWTH Aachen, 2002. (Cited on page 106.)

[LvHN05] Thorsten Liebig, Friedrich von Henke, and Olaf Noppens. Explanation support for OWL authoring. In *International Symposium on Explanation-aware Computing (ExaCt 2005)*, pages 86–93, 2005. (Cited on page 39.)

[MB95] Deborah L. McGuinness and Alex Borgida. Explaining subsumption in description logics. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 816–821, 1995. (Cited on pages 39 and 54.)

[McG96] Deborah L. McGuinness. *Explaining reasoning in description logics*. PhD thesis, Rutgers University, 1996. (Cited on pages 13 and 39.)

[MHL07] Yue Ma, Pascal Hitzler, and Zuoquan Lin. Algorithms for paraconsistent reasoning with OWL. In *Proceedings of the 4th European Semantic Web Conference (ESWC-07)*, pages 399–413, 2007. (Cited on page 30.)

[Mil56] George A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956. (Cited on pages 137 and 138.)

[Min74] Marvin Minsky. A framework for representing knowledge. MIT-AI Laboratory Memo 306, 1974. (Cited on page 22.)

[MIS12] Eleni Mikroyannidi, Luigi Iannone, and Robert Stevens. Rio: The regularities inspector for ontologies plugin for Protégé-4. In *ICBO*, 2012. (Cited on page 37.)

[MLBP06] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pages 269–274, 2006. (Cited on pages 41 and 43.)

[MLP06] Thomas Meyer, Kevin Lee, and Jeff Pan. Computing maximally satisfiable terminologies for the description logic $\mathcal{ALC}$ with cyclic definitions. 2006. (Cited on page 41.)

[MMIS12] Eleni Mikroyannidi, Nor Azlinayati Abdul Manaf, Luigi Iannone, and Robert Stevens. Analysing syntactic regularities in ontologies. In *Proceedings of the 9th International Workshop on OWL: Experiences and Directions (OWLED-12)*, 2012. (Cited on pages 33 and 37.)

[MMV10] Thomas Meyer, Kodylan Moodley, and Ivan Varzinczak. First steps in the computation of root justifications. In *Proc. of ARCOE-10*, 2010. (Cited on pages 53 and 176.)

[MSR11] Eleni Mikroyannidi, Robert Stevens, and Alan L. Rector. Identifying ontology design styles with metrics. In *7th International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2011. (Cited on pages 79 and 180.)

[NBH+06] Stephen E. Newstead, Peter Brandon, Simon J. Handley, Ian Dennis, and Jonathan St. B. T. Evans. Predicting the difficulty of complex logical reasoning problems. *Psychology Press*, 12:62–90, 2006. (Cited on page 131.)

[Neb90] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Springer-Verlag, 1990. (Cited on page 40.)

[NPPW12a] Tu Anh T. Nguyen, Richard Power, Paul Piwek, and Sandra Williams. Measuring the understandability of deduction rules for OWL. In *Proc. of WoDOOM-12*, 2012. (Cited on pages 52, 112, and 129.)

[NPPW12b] Tu Anh T. Nguyen, Richard Power, Paul Piwek, and Sandra Williams. Planning accessible explanations for entailments in OWL ontologies. In *Proceedings of the 7th International Natural Language Generation Conference (INLG 2012)*, pages 110–114, 2012. (Cited on page 52.)

[NRG12] Nadeschda Nikitina, Sebastian Rudolph, and Birte Glimm. Interactive ontology revision. *Journal of Web Semantics*, 12:118–130, 2012. (Cited on pages 55, 56, 74, and 133.)

[NSW+09] Natalya F. Noy, Nigam H. Shah, Patricia L. Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L. Rubin, Margaret-Anne Storey, Christopher G. Chute, and Mark A. Musen. Bioportal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 37:W170–W173, 2009. (Cited on page 33.)

[owl09] OWL 2 profiles. `http://www.w3.org/TR/owl2-profiles/` [Accessed: 10-10-2012], 2009. (Cited on page 32.)

[PS10] Rafael Peñaloza and Baris Sertkaya. Complexity of axiom pinpointing in the DL-Lite family of description logics. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, 2010. (Cited on page 41.)

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In *Proceedings of the 14th International World Wide Web Conference (WWW-05)*, pages 633–640, 2005. (Cited on page 16.)

[PSMB+91] Peter F. Patel-Schneider, Deborah L. McGuinness, Ronald J. Brachman, Lori Alperin Resnick, and Alexander Borgida. The CLASSIC

knowledge representation system: Guiding principles and implementation rationale. *SIGART Bulletin*, 2(3):108–113, 1991. (Cited on page 39.)

[Qui52] Willard Van Orman Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952. (Cited on page 69.)

[Qui59] Willard Van Orman Quine. On cores and prime implicants of truth functions. *The American Mathematical Monthly*, 66(9):755–760, 1959. (Cited on page 69.)

[RCVB09] Catherine Roussey, Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of OWL ontology antipatterns. In *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP-09)*, pages 205–206, 2009. (Cited on pages 36 and 129.)

[RDH+04] Alan L. 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. pages 63–81, 2004. (Cited on page 36.)

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. (Cited on pages 43 and 95.)

[RMC12] Mariano Rodriguez-Muro and Diego Calvanese. Quest, an OWL 2 QL reasoner for ontology-based data access. In *Proceedings of the 9th International Workshop on OWL: Experiences and Directions (OWLED-12)*, 2012. (Cited on page 32.)

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. (Cited on page 12.)

[RSFF12] Patrick Rodler, Kostyantyn M. Shchekotykhin, Philipp Fleiss, and Gerhard Friedrich. RIO: Minimizing user interaction in ontology debugging. *CoRR*, abs/1209.3734, 2012. (Cited on page 56.)

[Rud87] Piotr Rudnicki. Obvious inferences. *Journal of Automated Reasoning*, 3(4):383–393, 1987. (Cited on page 55.)

[SC03] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 355–362, 2003. (Cited on pages 13, 39, 43, and 135.)

[Sch05a] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *Proceedings of the 2nd European Semantic Web Conference (ESWC-05)*, pages 226–240, 2005. (Cited on pages 37, 53, and 96.)

[Sch05b] Stefan Schlobach. Diagnosing terminologies. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 670–675, 2005. (Cited on pages 41 and 43.)

[Sch11] Thomas Scharrenbach. *End-User Assisted Ontology Evolution in Uncertain Domains*. PhD thesis, University of Zurich, Faculty of Economics, 2011. (Cited on page 79.)

[SFJ08] Kostyantyn Shchekotykhin, Gerhard Friedrich, and Dietmar Jannach. On computing minimal conflicts for ontology debugging. In *Proceedings of the Workshop on Model-Based Systems (MBS-08)*, 2008. (Cited on page 43.)

[SFRF12] Kostyantyn M. Shchekotykhin, Philipp Fleiss, Patrick Rodler, and Gerhard Friedrich. Direct computation of diagnoses for ontology debugging. *CoRR*, abs/1209.0997, 2012. (Cited on pages 56 and 74.)

[SHCvH07] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39(3):317–349, 2007. (Cited on page 39.)

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A highly-efficient OWL reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED-08EU)*, 2008. (Cited on page 31.)

[Sow87] John F. Sowa. Semantic networks. In *Encyclopedia of Artificial Intelligence 2*. Journal of Web Semantics, 1987. (Cited on page 22.)

[SPC+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007. (Cited on page 31.)

[SQJH08] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A modularization-based approach to finding all justifications for OWL DL entailments. In *Proceedings of the 3rd Asian Semantic Web Conference (ASWC-08)*, pages 1–15, 2008. (Cited on page 44.)

[SS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991. (Cited on page 23.)

[SSZ09] Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which kind of module should I extract? In *Proceedings of the 22nd International Workshop on Description Logics (DL-09)*, 2009. (Cited on page 44.)

[ST99] Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999. (Cited on page 116.)

[Sun08] Boontawee Suntisrivaraporn. Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}+$ ontologies. In *Proceedings of the 5th European Semantic Web Conference (ESWC-08)*, pages 230–244, 2008. (Cited on pages 41 and 44.)

[TAM+05] Samir Tartir, I. Budak Arpinar, Michael Moore, Amith P. Sheth, and Boanerges Aleman-Meza. OntoQA: Metric-based ontology quality analysis. In *Proc. of KADASH-05*, 2005. (Cited on page 19.)

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR-06)*, 2006. (Cited on page 31.)

[Tuf] Edward Tufte. The magical number seven, plus or minus two: Not relevant for design. `http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0000U6` [Accessed: 11-01-2013]. (Cited on page 138.)

[W3C04a] W3C. RDF vocabulary description language 1.0: RDF schema. http://www.w3.org/TR/rdf-schema/, 2004. (Cited on page 30.)

[W3C04b] W3C. Resource description framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts/, 2004. (Cited on page 30.)

[W3C09] W3C. OWL 2 web ontology language. http://www.w3.org/TR/owl2-overview/, 2009. (Cited on pages 11 and 30.)

[W3C12] W3C. OWL 2 web ontology language RDF-based semantics (second edition). http://www.w3.org/TR/owl2-rdf-based-semantics/, 2012. (Cited on page 30.)

[WP06] Taowei David Wang and Bijan Parsia. CropCircles: Topology sensitive visualization of OWL class hierarchies. In *Proceedings of the 5th International Semantic Web Conference (ISWC-06)*, pages 695–708, 2006. (Cited on pages 57 and 179.)

[WPH06] Taowei David Wang, Bijan Parsia, and James Hendler. A survey of the web ontology landscape. In *Proceedings of the 5th International Semantic Web Conference (ISWC-06)*, pages 682–694, 2006. (Cited on page 12.)

Appendix A

Ontologies in the test corpus

ONT	EN	CJ	UC	DL	EL	QL	RL	CL	OP	DP	IN	DT	AX	DL
adverse event reporting	3	9	0	✓	✗	✗	✗	275	59	4	25	2	537	SROIQ
amino acid	112	2652	0	✓	✗	✗	✗	46	5	1	0	1	477	ALCF
basic formal ontology	116	459	0	✓	✗	✗	✗	39	0	0	0	0	95	ALC
basic vertebrate anatomy	292	7186	0	✓	✗	✗	✗	99	74	0	0	0	386	SHIF
bioassay	167	307	0	✗	✗	✗	✗	1293	120	13	45	4	1791	SROIQ
biological imaging methods	95	311	0	✓	✓	✗	✗	517	1	0	0	0	548	EL++
biopax	10	12	0	✗	✗	✗	✗	70	55	41	0	5	399	SHIN
bleeding history phenotype	421	1313	0	✓	✗	✗	✗	544	33	5	0	2	1925	ALCIF
bone dysplasia	800	1114	0	✓	✗	✗	✗	13839	28	6	9	3	44683	SHIF
breast cancer grading	109	109	0	✗	✗	✗	✗	134	56	27	193	6	1034	SHOIN
cancer research	10	378	0	✓	✗	✗	✗	1770	242	15	61	8	5522	SROIQ
cao	767	5800	0	✓	✗	✗	✗	204	35	2	0	2	442	SHIQ
cereal plant gross anatomy	250	729	0	✓	✓	✗	✗	1181	3	0	0	0	2091	EL++
cerebrotendinous xanth.	14	15	0	✗	✗	✗	✗	292	22	11	282	6	1972	ALCOIN
chemical information	132	1962	4	✓	✗	✗	✗	579	59	6	22	9	1230	SHOIN
cognitive atlas	146	810	0	✓	✗	✗	✗	1701	6	5	1697	1	8791	ALC
cognitive paradigm	406	4252	0	✓	✗	✗	✗	355	49	4	22	2	682	SHOIN
comparative data analysis	121	532	1	✗	✗	✗	✗	127	74	11	8	7	813	SROIQ
computational neuroscience	20	86	0	✓	✗	✗	✗	214	10	4	165	2	1241	ALCOI
dendritic cell	142	143	0	✓	✗	✗	✗	148	9	0	0	0	313	ALC
dikb evidence	113	124	0	✗	✗	✗	✗	124	70	37	8	6	645	ALCHOIN
eagle i research resource	662	12227	0	✓	✗	✗	✗	2239	103	54	22	2	3114	SHOIF
electrocardiography	37	73	0	✓	✗	✗	✗	1098	20	24	0	1	1274	ALCIF
emotion	423	1558	0	✓	✗	✗	✗	404	60	4	20	2	698	SHOIQ
environment	69	138	0	✓	✓	✗	✗	1538	7	0	0	0	1807	EL++
enzyme mechanism	19	104	0	✓	✗	✗	✗	260	32	1	50	1	931	ALCRQ
evidence codes	441	662	0	✓	✓	✗	✗	268	1	0	0	0	345	EL++

Table A.1: ONT = ontology name, EN = # entailments, CJ = # complex justifications, UC = # unsat. classes, DL/EL/QL/RL = OWL 2 profile, CL = # classes, OP = # object properties, DP = # data properties, IN = # individuals, DT = # datatypes, AX = # logical axioms, EX = DL expressivity

ONT	EN	CJ	UC	DL	EL	QL	RL	CL	OP	DP	IN	DT	AX	DL
family health history	522	665	0	✗	✗	✗	✗	239	431	1	12	1	1111	ALCHIF
fission yeast phenotype	770	10630	0	✓	✓	✗	✗	1571	20	0	0	0	2257	EL++
gene ontology extension	32	6421	0	✓	✓	✗	✗	32841	6	0	0	0	68777	EL++
gene regulation	564	1167	0	✗	✗	✗	✗	508	24	5	4	2	962	ALCHIQ
general formal ontology	16	21	0	✓	✗	✗	✗	45	41	0	0	0	212	SHIQ
host pathogen interactions	669	4056	0	✓	✗	✗	✗	278	30	0	0	0	403	SHI
imgt	4	23	0	✓	✗	✗	✗	292	23	4	0	1	2114	SHIN
infectious disease	821	7570	0	✓	✗	✗	✗	509	41	0	19	0	1245	SROIF
information artifact	218	1149	0	✓	✗	✗	✗	116	36	1	33	1	314	SHOIN
int. class. nursing practice	762	863	0	✓	✗	✗	✗	3290	33	0	0	0	11891	SHIF
interaction network	293	1063	0	✓	✗	✗	✗	978	0	0	0	0	1034	ALC
kinetic simulation algorithm	202	2617	0	✓	✗	✗	✗	224	9	2	0	4	706	ALCRIQ
lipid	665	4724	0	✓	✗	✗	✗	716	46	0	0	0	2375	ALCHIN
mahco an mhc	178	2279	0	✓	✗	✗	✗	7929	8	3	0	1	13781	ALCIQ
mged	85	161	0	✗	✗	✗	✗	237	81	45	713	5	1478	ALEOF
nanoparticle	43	633	0	✓	✗	✗	✗	1816	50	8	0	2	16267	SHIN
neomark oral cancer	505	2365	0	✓	✗	✗	✗	325	26	0	0	0	399	SHIQ
neural electromagnetic	238	1338	0	✓	✗	✗	✗	1633	89	4	56	3	2681	SHIQ
nif dysfunction	461	1790	0	✗	✗	✗	✗	1860	44	12	178	2	2696	SROIF
nmr instrument specific comp.	423	3119	0	✓	✗	✗	✗	481	9	9	32	2	676	SH
non random. controlled trials	530	2275	5	✓	✗	✗	✗	156	4	5	31	3	508	ALCOF
o. for drug discovery investig.	1	38	0	✓	✗	✗	✗	659	60	4	20	2	1001	SHOIN
o. for general medical science	709	14	0	✓	✗	✗	✗	133	0	0	19	0	216	ALCO
o. for genetic interval	224	7	0	✗	✗	✗	✗	212	51	14	22	4	524	SHIN
o. for microrna target pred.	84	1	0	✓	✗	✓	✗	320	16	2	0	2	415	ALCI
o. of adverse events oae	228	6993	0	✓	✗	✗	✗	653	28	0	0	0	807	SHI

Table A.1: ONT = ontology name, EN = # entailments, CJ = # complex justifications, UC = # unsat. classes, DL/EL/QL/RL = OWL 2 profile, CL = # classes, OP = # object properties, DP = # data properties, IN = # individuals, DT = # datatypes, AX = # logical axioms, EX = DL expressivity

ONT	EN	CJ	UC	DL	EL	QL	RL	CL	OP	DP	IN	DT	AX	DL
o. of clinical research ocre	208	1163	0	✗	✗	✗	✗	13	13	2	0	1	51	ALCHIF
o. of exp. variables and values	1	1147	0	✓	✗	✗	✗	39	22	20	147	5	487	ALCO
o. of physics for biology	100	9832	0	✓	✗	✗	✗	297	5	3	0	1	447	ALCHIQ
oboe	36	780	0	✓	✗	✗	✗	40	32	10	0	4	265	SRIQ
oboe sbc	12	4	0	✓	✗	✗	✗	629	26	6	0	4	1359	SRIQ
ontologia proj alternativa	4	747	0	✗	✗	✗	✗	22	20	68	2	5	288	ALUIN+
parasite experiment o.	187	4064	0	✓	✗	✗	✗	144	26	19	27	5	348	ALCHQ
pathogen transmission	1	1	0	✓	✓	✓	✓	25	0	0	0	0	24	EL++
pharmacovigilance	140	140	0	✓	✓	✓	✓	33	0	0	0	0	24	EL++
phenotypic quality	18	44	0	✓	✓	✗	✗	1404	12	0	0	0	1884	EL++
phenx toolkit	295	1354	0	✓	✓	✓	✓	4	0	0	2	0	5	EL++
plant environ. conditions	15	45	0	✓	✓	✓	✓	483	0	0	0	0	499	EL++
plant growth and dev. stage	107	379	0	✓	✓	✗	✗	239	1	0	0	0	240	EL++
protein modification	165	374	0	✓	✓	✗	✗	1313	4	0	0	0	1986	EL++
smoking behavior risk	290	1971	0	✓	✗	✗	✗	121	12	0	0	0	185	ALEI+
snp	182	935	0	✗	✗	✗	✗	2258	101	12	4	3	11199	SHOIN
solanaceae phenotype	584	336	0	✓	✓	✓	✗	380	1	0	0	0	411	EL++
soyontology	645	940	0	✓	✓	✓	✓	1817	0	0	0	0	1816	EL++
student health record	196	255	0	✓	✗	✗	✗	344	36	2	0	2	418	ALH
synapse	508	10624	0	✓	✗	✗	✓	6464	1	0	0	0	6743	ALF
teleost taxonomy	2	4	0	✓	✓	✓	✓	38640	0	0	0	0	38639	EL++
translational medicine	216	1697	0	✓	✗	✗	✗	300	33	6	15	6	502	SRIN
vertebrate anatomy	528	2867	0	✓	✓	✗	✗	230	13	0	0	0	392	EL++
vivo	168	907	0	✓	✗	✗	✗	217	220	178	388	8	1712	ALEHIN+
zebrafish anat. and dev.	20	32	0	✓	✓	✗	✗	2740	5	0	0	0	10495	EL++

Table A.1: ONT = ontology name, EN = # entailments, CJ = # complex justifications, UC = # unsat. classes, DL/EL/QL/RL = OWL 2 profile, CL = # classes, OP = # object properties, DP = # data properties, IN = # individuals, DT = # datatypes, AX = # logical axioms, EX = DL expressivity