JUSTIFICATION BASED EXPLANATION IN ONTOLOGIES

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

2011

By
Matthew Horridge
School of Computer Science

Contents

Number of Words: 67,078

List of Tables

List of Figures

Abstract

The Web Ontology Language, OWL, is the latest standard in logic based ontology languages. It is built upon the foundations of highly expressive Description Logics, which are fragments of First Order Logic. These logical foundations mean that it is possible to compute what is entailed by an OWL ontology. The reasons for entailments can range from fairly simple localised reasons through to highly non-obvious reasons. In both cases, without tool support that provides explanations for entailments, it can be very difficult or impossible to understand why an entailment holds. In the OWL world, justifications, which are minimal entailing subsets of ontologies, have emerged as the dominant form of explanation.

This thesis investigates justification based explanation techniques. The core of the thesis is devoted to defining and analysing Laconic and Precise Justifications. These are fine-grained justifications whose axioms do not contain any superfluous parts. Optimised algorithms for computing these justifications are presented, and an extensive empirical investigation shows that these algorithms perform well on state of the art, large and expressive bio-medical ontologies. The investigation also highlights the prevalence of superfluity in real ontologies, along with the related phenomena of justification masking. The practicality of computing Laconic Justifications coupled with the prevalence of non-laconic justifications in the wild indicates that Laconic and Precise justifications are likely to be useful in practice.

The work presented in this thesis should be of interest to researchers in the area of knowledge representation and reasoning, and developers of reasoners and ontology editors, who wish to incorporate explanation generation techniques into their systems.

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

Acknowledgements

First and foremost, I would like to thank my supervisors: Bijan Parsia and Uli Sattler. I would never have completed a PhD without them and I could not have asked for better supervisors.

I would like to thank Alan Rector for being my PhD advisor and for always finding funding that enabled me to attend numerous conferences and workshops.

I would like to thank the members of the Bio-Health Informatics Group and Information Management Group in Manchester, in particular Simon Harper, Ignazio Palmisano and Thomas Schneider. Special thanks are due to Dmitry Tsarkov for always showing a keen interest in my work and for the many technical discussions that we had throughout my time as a PhD student.

Finally, I would like to thank my family and Simon for always being there for me and putting up with me while I finished this thing.

Chapter 1

Introduction

An ontology is a machine processable artefact that captures the relationships between concepts and objects in some domain of interest. In 2004 the Web Ontology Language, OWL [HPSvH03], became a World Wide Web Consortium Standard. Since then, it has become the most widely used ontology language, being adopted all over the world by academia and industry alike. As well as being extensively used in the field of computer science, OWL is used by many domain experts in diverse areas such as bioinformatics, medicine and geography.

One of the key aspects of OWL is that it is built upon the foundations of a *Description Logic*. Description logics [BCM$^+$03] are a family of knowledge representation languages which are typically *decidable fragments* of First Order Logic. This means that each OWL ontology may be viewed as a set of Description Logic *axioms*, or first order sentences, and is therefore a logical theory. This logical foundation gives statements made in OWL a well defined, unambiguous meaning. Moreover, it makes it possible to specify various automated reasoning tasks, and to use "off the shelf" description logic reasoners such as FaCT++ [TH06], HermiT [MSH07], Pellet [SPG$^+$07], Racer [HM01], CB [Kaz09] and CEL [BLS06a] for computing relationships between the various concepts and objects that are expressed in an ontology. With reasoning, knowledge that was implicit, but not stated in the original ontology, can be made explicit—knowledge can be *inferred.*

A consequence of the fact that OWL corresponds to a highly expressive description logic is that unexpected and undesirable inferences (entailments), can arise during the construction of an ontology. The reasons as to why an entailment holds in an ontology can range from simple localised reasons through to highly non-obvious reasons. In the case of very large ontologies such as the National

Cancer Institute Thesaurus [GFH+03], which contains over 80,000 axioms, or the large medical ontology SNOMED [SC97], which contains over 500,000 axioms, manually pinpointing the reasons for an entailment can be a wretched and error prone task. Without automated explanation support it can be very difficult to track down the axioms in an ontology that give rise to entailments. It is for this reason that explanation is an important topic in this area.

Indeed, since OWL became a standard, there has been a large demand from ontology developers for tools that can provide explanations for entailments. Some common tasks that prompt a demand for explanation facilities are:

1. **Understanding entailments**—A user browsing an ontology notices an entailment and opportunistically decides to obtain an explanation for the entailment in order to find out what has been stated in the ontology that causes the entailment to hold.

2. **Debugging and repair of ontologies**—A user is faced with an incoherent or inconsistent ontology, or an ontology that contains some other kind of undesirable entailment. They need to determine the causes of the entailment in order to understand why it holds so that they can generate a repair plan.

3. **Ontology comprehension**—A user is faced with an ontology that they have not seen before. In order to get a better picture of the ontology they use various metrics such as the number of entailments, the average number of explanations for an entailment and so on. This helps them to build up an image of how complex the ontology is in terms of expressivity. Is also provides them with more information if they need to decide whether they like the ontology or not.

1.1 Justification Based Explanation

In the world of OWL and Description Logics there has been a significant amount of research devoted to the area of explanation and ontology debugging. In particular, research has focused on a specific type of explanation called *justifications* [BH95, SC03, Kal06, BPS07]. A justification for an entailment in an ontology is a minimal subset of the ontology that is sufficient for the entailment to hold. The set of axioms corresponding to the justification is minimal in the sense that if an axiom is removed from the set, the remaining axioms no longer support the entailment. More precisely, given an ontology $\mathcal{O}$ and an entailment η such that $\mathcal{O} \models \eta$, $\mathcal{J}$ is a justification for η in $\mathcal{O}$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subsetneq \mathcal{J}$ it is the case that $\mathcal{J}' \not\models \eta$.

Justifications have turned out to be a very attractive form of explanation: They are conceptually simple, they have a clear relationship with the ontology from which they are derived, there are off-the-shelf algorithms for computing them, and there are simple presentation strategies which work well most of the time. Since justifications came to prominence in the ontology browsing and editing tool Swoop [KPH05], several other major open source and commercial ontology editing environments such as Protégé-4 [KFNM04], The NeOn Toolkit [JHQ+09], Top Braid Composer [Top09] and OWLSight [Gro09] have also adopted them for explaining entailments in ontologies. In essence, over the last few years, the use of justifications has advanced debugging and explanation support for entailments in OWL ontologies from virtually nothing to being respectable.

1.2 The Rise to Prominence of Justifications

Explanation has long been recognised as a key component in knowledge representation systems [MPS98, BHGS01, Hor05, LBF+06]. One of the most prominent early DL systems to feature an explanation component was the CLASSIC [BBMR89] system, where explanation was recognised as being very important for the usability of the system [MPS98]. In this early work, an explanation was essentially regarded as a proof, or a fragment of a proof, which explained how a reasoner proved that an entailment held in some ontology. In fact, there was a general feeling that an explanation system had to be closely allied with the reasoning system that proved entailments [McG96], and that proof based explanations were essentially *declarative* views on the structural reasoning procedures that were used at the time. This was a point of view that was maintained when more expressive Description Logics, such as $\mathcal{ALC}$ [SSS91], which featured sound and complete tableau-based reasoning procedures [BS01], started to come to the fore. Indeed, the notion of proof-based explanations was defined for $\mathcal{ALC}$ in [BFH00], extended and implemented in one form or another in [Kwo05] and [LN04], and also relatively recently adapted to the DL-Lite [CLLR05] family of

Description Logics in [BCR08].

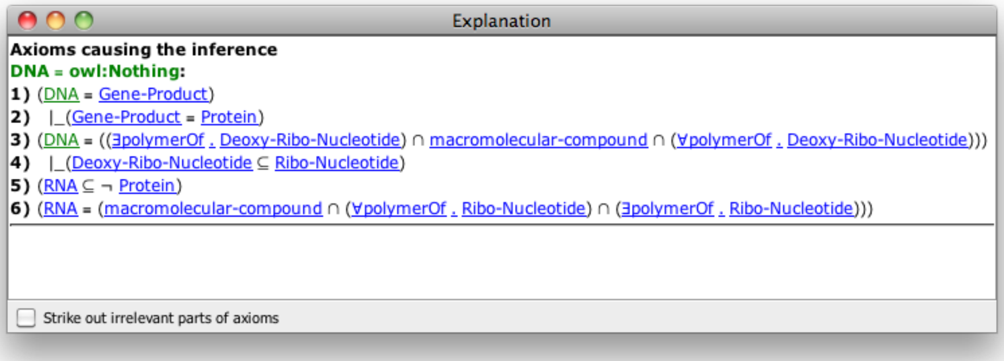
1.2.1 From Proofs to Justifications

While the ideas of proof based explanations in Description Logics are still around, the years between 2003 and 2005 marked a turning point in explanation for Description Logic and OWL based systems. Specifically, the fundamental idea of what constituted an explanation completely changed. This paradigm shift was centred around two seminal pieces of work: The first by Schlobach and Cornet [SC03] in 2003, and the second by Parsia, Kalyanpur et al. [PSK05] in 2005.

In [SC03], Schlobach and Cornet presented work on diagnosing and repairing ontologies that contained unsatisfiable concepts. Their work, part of which turned out to be closely related to early work by Baader and Hollunder [BH95], was motivated by the DICE ontology [dKAHC$^+$99, dKBRJC08] which is a large Description Logic based ontology for intensive care. Most importantly, Schlobach and Cornet introduced the notion of *Minimal Unsatisfiable Preserving Sub-TBoxes* (MUPS). These are minimal subsets of an ontology that are sufficient for entailing the unsatisfiability of a particular concept, and in essence are justifications for unsatisfiable classes.

In 2004, the Web Ontology Language OWL became a W3C recommendation (standard). However, people had already begun to build OWL ontologies before this. First using early editors such as OilEd [BHGS01], which were originally built for editing DAML+OIL [Hor02] ontologies—a precursor to OWL, and then using early versions of Protégé-OWL and Swoop. During this time, despite earlier work on proof based explanation, it became apparent that the tools and techniques for debugging inconsistent ontologies, or ontologies containing unsatisfiable classes, were non-existent. Ontology developers used trial and error to resolve problems, essentially ripping out axioms from their ontologies until classes turned satisfiable or the ontologies turned consistent. The only indications and debugging cues that were available were error messages saying that an ontology was inconsistent, or class names were painted in red to indicate that classes were unsatisfiable. It was obvious to those in the area that some sort of automated debugging support was desperately needed.

It was at this time that Parsia and Kalyanpur began to investigate techniques for the debugging and repair of OWL ontologies. In [PSK05], and then



Explanation

Axioms causing the inference

DNA ≡ owl:Nothing:

```
1) (DNA ≡ Gene-Product)
2)   |_(Gene-Product ≡ Protein)
3) (DNA ≡ ((∃polymerOf . Deoxy-Ribo-Nucleotide) ∩ macromolecular-compound ∩ (∀polymerOf . Deoxy-Ribo-Nucleotide)))
4)   |_(Deoxy-Ribo-Nucleotide ⊆ Ribo-Nucleotide)
5) (RNA ⊆ ¬ Protein)
6) (RNA ≡ (macromolecular-compound ∩ (∀polymerOf . Ribo-Nucleotide) ∩ (∃polymerOf . Ribo-Nucleotide)))
```

☐ Strike out irrelevant parts of axioms

Figure 1.1: An example of justification presentation in Swoop

in subsequent publications [KPSH05, KPS05, KPSG06], they introduced various important OWL ontology debugging techniques. Ultimately, Kalyanpur and Parsia were responsible for seeing the value of justifications as explanation and debugging devices and bringing justification based explanation to the masses in Swoop [KPH05]. This work culminated in Kalyanpur's PhD thesis [Kal06] which brought the notions of justifications, root unsatisfiable classes[1], and justification based repair together and demonstrated the overwhelming benefit of justification-based debugging support for repairing ontologies.

In the space of three years explanation had shifted from proof based explanation to justification based explanation—gone were the inference rules and steps that characterise proofs, along with the close tie to any particular proof procedure—justifications had taken their place.

1.2.2 Justification Based Explanation Today

In years following both Schlobach's and then Parsia and Kalyanpur's work there has been a huge interest in this area, and other researchers began to work on methods and techniques for computing and working with justifications [LMPB06, MLBP06, KK07, BPS07, BS08, KPHS07, Lam07]. Such was the demand for explanation by people developing and working with ontologies, many of the major ontology development environments began to offer justifications as a popular form

[1]A root unsatisfiable class is a concept name, in the signature of an ontology, whose unsatisfiability does not depend upon the unsatisfiability of some other concept name in the signature of that ontology—see Definition 2 in Section 2.3.2 on Page 46

of explanation. Today, virtually all of the major open source OWL ontology development tools, for example, Protégé-4 [KFNM04, HTR06], Swoop [KPH05] (as shown in Figure 1.1), and The NeOn Toolkit [JHQ+09], along with some commercial offerings such as Top Braid Composer [Top09] and OWLSight [Gro09] all feature the ability to compute and display justifications.

While the primary use of justifications is still explanation, people are also increasingly using them to get a feel of the "shape" of an ontology. In this case, when people come across an arbitrary entailment, they request a justification for the entailment so as to get a feel as to what kinds of axioms and constructs in the ontology give rise to the entailment.

1.2.3 Justifications in Auxiliary Services

Justifications are also increasingly being used for purposes other than explanation. For example, they are used for enriching ontologies [Sch05a], belief base revision [HWKP06], scalable ABox reasoning [DFK+07], incremental reasoning [CHWK07], reasoner completeness testing [SCH10], meta-modelling support [GRV10], default reasoning [SdF+10] and elimination of redundant axioms in ontologies [GW11]. There is plenty of evidence that they have utility within the OWL and Description Logic communities beyond the starting point of explanation.

1.2.4 Justifications in Other Fields

Finally, although Schlobach and Cornet were the first to introduce minimal entailing sets of axioms as forms of explanation to the Description Logic community, the same notion is also used in other communities. For example, *minimal conflict sets* (MCSs) in the area of model based diagnosis [Rei87, HCK92] correspond to justifications. Similarly, *irreducible inconsistent systems* (IISs) [Chi97] in the area of linear programming also correspond to justifications. Lastly, the Propositional Logic reasoning community use Minimal Unsatisfiable Sub-formulae (MUSes) for explaining why a set of clauses is unsatisfiable [BS05].

In summary, justifications are a widely used form of explanation. They dominate the current crop of tools and techniques for debugging and repairing ontologies, and they are widely used for purposes other than explanation.

1.3 Topics Covered in this Thesis

Despite the popularity of justifications as forms of explanation, there are several unresolved issues with them. This thesis deals with three issues that are related to each other and are outlined below. It should be noted that the related work on each topic has been localised to the relevant chapters and the purpose of what follows is to provide a flavour of what is to come.

1.3.1 Computing Justifications

The use of justifications in real applications depends on having practical algorithms that perform well on realistic inputs. Indeed, much of the work presented in this thesis, and the performance of tools that provide explanation support or services which use justification finding under the hood, hinges on the ability to compute justifications for entailments in real ontologies. The basic technique of finding justifications for an entailment in a set of axioms is required over and over again. Despite this, there is a lack of strong convincing evidence that shows whether or not computing justifications for entailments in real ontologies is practical. Prior to the work presented here, testing has mainly focused on how one optimisation performs over another and has typically been carried out on test-bed ontologies with prototypical implementations. The work presented in Chapters 3–6 therefore focuses on computing justifications. It uses a large corpus of published ontologies to thoroughly test robustly implemented justification finding algorithms.

1.3.2 Justification Granularity

Justifications operate at the level of asserted axioms. That is, they are composed of the statements that people write down in their ontologies. In many ontologies axioms are built up of nested complex concepts and can be large with many parts. This means that, as far as justifications are concerned, there can be parts of axioms that are superfluous for the entailment in question. These parts, superfluous or otherwise, can cause various usability problems including problems associated with understanding and problems associated with repair.

Justifications whose axioms do not contain any superfluous parts have been referred to as *fine-grained justifications* [LSPV08] and *precise justifications* [KPG06].

However, there is no definitive answer as to what a fine-grained or precise justification is. Indeed, there have been no thorough investigations into these kinds of justifications, their properties and general methods of computing them. This thesis therefore focuses on producing a formal definition of fine-grained justifications, methods of computing them, and an evaluation of these methods on a large ontology corpus which reveals the fine-grained justification landscape.

1.3.3 Understanding Justifications

The last part of the thesis focuses on understanding justifications and looking at mechanisms which can be used to make difficult justifications easier to understand. While the use of justifications is widespread, and most of the time they seem to serve well as a form of explanation, there is anecdotal evidence to suggest that some naturally occurring justifications can be difficult or impossible to understand. In essence, justifications are merely the premises of a proof and, as such, do not articulate the sometimes non-obvious reasoning steps which connect those premises with the conclusion. While it is seems unlikely that full blown proofs are necessary, it does seem like it would be fruitful to explore the issue of justification understanding and suggest mechanisms that could be used to cope with difficult to understand justifications. This thesis therefore presents several user studies that were carried out in order to investigate these issues, and it takes a speculative look at potential mechanisms for coping with the problem of justification understanding.

1.4 Contributions of this Thesis

In summary, justifications have become the dominant form of explanation in OWL and in Description Logics. Therefore, the broad aims of this thesis are to advance the state of the art in knowledge about computing justifications, introduce definitions of fine-grained justifications and methods of computing them, and to look at the explanatory power and understandability of justifications. In doing this the thesis makes the following contributions:

1. It provides strong evidence in the form of a thorough empirical evaluation which shows that computing all justifications for entailments in realistic ontologies is practical.

2. It presents the first proper definition for fine-grained justifications and introduces two new types of justifications: Laconic and Precise Justifications, which avoid some of the problems associated with the level of granularity of regular justifications.

3. It provides optimised algorithms for computing Laconic and Precise Justifications, and carries out a thorough evaluation which shows that it is practical to compute all preferred Laconic Justifications for entailments in realistic ontologies.

4. It provides strong evidence of need for Laconic and Precise Justifications in the form of results which show that superfluity is common throughout real ontologies, and highlights some of the issues with regular justifications due to the fact that they do not pay attention to parts of axioms.

5. It examines the understandability of justifications and concludes that there are naturally occurring justifications which are difficult or impossible for many people, including those with a significant level of experience in OWL, to understand.

6. It proposes a model for predicting the complexity of justifications and a methodology for refining and evolving the model.

7. It proposes complexity model driven justification lemmatisation and justification oriented proofs as mechanisms for introducing helpful intermediate inference steps into justifications that are difficult to understand.

1.5 Published Work

The work embodied in this thesis is supported by several workshop and conference publications:

Chapters 3–6: Computing Justifications

- [KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In Karl Aberer, Key-Sun Choi, Natalya F. Noy, Guus Schreiber, and Riichiro Mizoguchi, editors, *The Semantic Web - 6th International Semantic Web Conference,* *2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2007

- [HPS09a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Computing explanations for entailments in Description Logic based ontologies. In *16th Automated Reasoning Workshop (ARW 2009), Liverpool, UK.*, 2009

- [HPS09b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explaining inconsistencies in OWL ontologies. In Luis Godo and Andrea Pugliese, editors, *3rd International Conference on Scalable Uncertainty Management SUM 2009, September 28–30, 2009 Washington DC Area, USA*, volume 5785 of *Lecture Notes In Computer Science*, pages 124–137. Springer, 2009

- [HPS11] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The state of biomedical ontologies. In *BioOntologies 2011 Co-Located with ISMB 2011, 15th–16th July, Vienna Austria*, 2011

Chapters 7–11: Laconic and Precise Justifications

- [HPS08b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web – ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008.ISWC 2008*, volume 5318 of *Lecture Notes In Computer Science*, pages 323–338. Springer, October 2008

- [HBPS08] Matthew Horridge, Johannes Bauer, Bijan Parsia, and Ulrike Sattler. Understanding entailments in OWL. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2008

- [HPS10a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification masking in OWL. In Grant Weddell, Volker Haarslev, and David Toman, editors, *Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Canada. May 4th–May 7th, 2010*, 2010

Chapters 12–13: Understanding Justifications

- [HPS08a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explanation of OWL entailments in Protégé-4. In Christian Bizer and Anupam Joshi, editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2008

- [HPS09a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Computing explanations for entailments in Description Logic based ontologies. In *16th Automated Reasoning Workshop (ARW 2009), Liverpool, UK.*, 2009

- [HPS09d] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in OWL. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics (DL 2009)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, July 2009

- [HPS09c] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. From justifications to proofs for entailments in OWL. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009

- [HPS09c] Matthew Horridge and Bijan Parsia. From justifications towards proofs for ontology engineering. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010

- [HPS10b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification oriented proofs in OWL. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes In Computer Science*, pages 354–369. Springer, November 2010

- [HBPS11a] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyashev, editors, *Proceedings of the 24th International Workshop on Description Logics (DL2011), Barcelona, Spain July 13–16, 2011*, CEUR Workshop Proceedings. CEUR-WS.org, July 2011

- [HBPS11b] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In Christopher A. Welty, Lora Aroyo, and Natalya F. Noy, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, ISWC 2011, Bonn, Germany, October 23-27, 2011*, Lecture Notes In Computer Science. Springer, 2011

Chapter 2

Preliminaries

This chapter introduces the nomenclature, key ideas and definitions that are used in the following chapters. The work presented in this thesis focuses on justification based explanation techniques for entailments in ontologies written in the $\mathcal{SHOIQ}$ [HS07] Description Logic. This is an expressive Description Logic which is slightly more expressive than the $\mathcal{SHOIN}$ Description Logic that underpins OWL, but not as expressive a $\mathcal{SROIQ}$ [HKS06] which underpins OWL 2 [MPSP09, CHM+08].

2.1 Basic Description Logics

Description Logics (DLs) are a family of logic based knowledge representation languages [BCM+03]. Generally speaking, they are *decidable* fragments of First Order Logic. Two features that distinguish them from First Order Logic are: (1) There are usually practical decision procedures for key inference problems; (2) They have their own special variable free syntax that is more concise than First Order Logic syntax and is particularly geared towards providing high level model primitives.

Concepts, Roles and Individuals

There are a large number of well studied DLs. Each of these is defined by the modelling primitives that it admits. Whatever the full set of modelling primitives, most DLs have the basic building blocks of *concepts*, *roles* and *individuals*:

- **Concepts** correspond to unary predicates in First Order Logic, and are used to describe classes of objects that share common properties and characteristics. In OWL concepts are known as *classes*. Typical examples of *named* concepts are Person, Father, Man and Woman. In this thesis, the letters A and B, and X sometimes augmented with subscripts, are used to denote concept names.

- **Roles** correspond to binary predicates in First Order Logic, and are used to describe the relationships between objects. In OWL, roles are known as *properties*. $\mathcal{SHOIQ}$ only supports binary roles, but some description logics, for example $\mathcal{DLR}$ [CGL97], allow n-ary roles. Typical examples of roles are hasChild, hasMother and hasParent. In this thesis the letters R and S are used to denote roles.

- **Individuals** correspond to constants in First Order Logic, and are used to represent single objects in the domain of interest. Examples of individuals are Matthew, Manchester and TheUniversityOfManchester. Individuals are said to be *instances* of classes. In this thesis, the lower case letters a, b and o are used to denote individuals.

Complex Concepts

Description Logics allow complex concepts to be composed from other concepts. This is done via the use of *concept constructors*. A given Description Logic supports a given set of concept constructors and is usually characterised by this set. For example, the Description Logic $\mathcal{ALC}$ supports concept *intersection* ($\sqcap$), concept *union* ($\sqcup$), concept *negation* ($\neg$), *existential quantification* ($\exists$) and *universal quantification* ($\forall$). For example, the concept $\mathsf{Mother} \sqcap \mathsf{AirlinePilot}$ describes the individuals that are both instances of Mother *and* AirlinePilot. The concept $\mathsf{PrivatePilot} \sqcup \mathsf{AirlinePilot}$ describes individuals that instances of PrivatePilot *or* AirlinePilot (or both). The concept $\neg\mathsf{Pilot}$ describes the individuals that are instances of the complement of Pilot i.e. not Pilot. The concept $\exists\mathsf{hasSibling}.\mathsf{AirlinePilot}$ describes the individuals that have at least one hasSibling relationship to some instance of the concept AirlinePilot. Finally, the concept $\forall\mathsf{hasSibling}.\mathsf{CommercialPilot}$ describes the individuals that either have no hasSibling relationships at all, or *only* have hasSibling relationships to instances of CommercialPilot.

It is usually the case that, concepts can be *nested* to arbitrary levels, so for example, the concept $\mathsf{Person} \sqcap (\exists\mathsf{hasSister}.(\mathsf{Pilot} \sqcup \mathsf{Engineer}))$ describes the individuals that are instances of Person *and* have at least one hasSister relationship to an individual that is either a Pilot or an Engineer (or both).

In this thesis, the letters C and D, possibly augmented with subscripts, are used to denote possibly complex concepts.

Axioms

An axiom is a statement, that specifies how the concepts, roles and individuals in the domain of interest relate to each other. At a high level, axioms are categorised into TBox axioms, RBox axioms and ABox axioms.

- **TBox axioms** specify relationships between concepts, and are of the form $C \sqsubseteq D$, which is read as "C subclass of D". They are known as *General Concept Inclusions* (GCIs), *concept subsumption axioms* or *subclass axioms* in OWL. For example, the axiom AirlinePilot $\sqsubseteq$ CommercialPilot states that the concept AirlinePilot is a *subconcept of* CommercialPilot. That is, every instance of AirlinePilot is also an instance of CommercialPilot. The axiom AirlinePilot $\sqsubseteq$ $\exists$hasLicense.AirTransportPilotsLicense states that every instance of AirlinePilot has an air transport pilot's license. An abbreviation for the pair of axioms $C \sqsubseteq D$ and $D \sqsubseteq C$ is $C \equiv D$, which reads as "C is equivalent to D".

- **RBox axioms** specify relationships between roles. The most basic RBox axioms are of the form $R \sqsubseteq S$, which reads as "R is a subrole of S". They are known as *Role Inclusion Axioms*, *role subsumption axioms* or *sub-property axioms* in OWL. For example, the axiom hasSister $\sqsubseteq$ hasSibling states that hasSister is a *sub-role of* hasSibling. That is, every pair of individuals in a hasSister relationship is also in a hasSibling relationship.

- **ABox axioms** specify relationships between concepts and individuals, and individuals and roles. The most basic ABox axioms are of the form $C(a)$ and $R(a, b)$, which are known as *concept assertions* and *role assertions* respectively. For example, AirlinePilot(John) states that the individual John is an *instance* of the concept AirlinePilot. The ABox axiom hasSister(John, Mary) states that John is related to Mary via the hasSister role.

The kinds of axioms available depend upon the description logic in question, but most, if not all, Description Logics feature concept-subsumption axioms between concept names (and possibly complex concepts).

Constructor	Syntax	Example
Top concept	$\top$	
Bottom concept	$\bot$	
Complex concept negation	$\neg C$	$\neg\mathsf{Human}$
Concept intersection	$C \sqcap D$	$\mathsf{Mother} \sqcap \mathsf{Pilot}$
Concept union	$C \sqcup D$	$\mathsf{AirlinePilot} \sqcup \mathsf{FighterPilot}$
Existential restriction	$\exists R.C$	$\exists\mathsf{hasSibling}.\mathsf{AirlinePilot}$
Universal restriction	$\forall R.C$	$\forall\mathsf{hasSibling}.\mathsf{FighterPilot}$

Table 2.1: $\mathcal{ALC}$ concept constructors and examples

2.1.1 The $\mathcal{ALC}$ Description Logic

As mentioned previously, a particular Description Logic is characterised by the concept constructors, role constructors and types of axioms that it admits. One of the first Description Logics to be studied in depth, and which is regarded as a prototypical Description Logic, is $\mathcal{ALC}$ [SSS91].

$\mathcal{ALC}$ supports concept intersection ($\sqcap$), concept union ($\sqcup$), negation of complex concepts ($\neg$), existential restriction ($\exists$) and universal restriction ($\forall$). For a concept name A, and a role name R, $\mathcal{ALC}$ concepts are defined as follows, where C and D are themselves $\mathcal{ALC}$ concepts,

$$\top \mid \bot \mid A \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \exists R.C \mid \forall R.C$$

The names and syntax of $\mathcal{ALC}$ concepts, along with examples are shown in Table 2.1. $\mathcal{ALC}$ axioms are either concept subsumption axioms of the form $C \sqsubseteq D$, concept equivalence axioms of the form $C \equiv D$, where C and D are $\mathcal{ALC}$ concepts, or individual assertions of the form $C(a)$ or $R(a, b)$, where C is an $\mathcal{ALC}$ concept and a and b are individuals.

Two Description Logics that are slightly less expressive than $\mathcal{ALC}$ are obtained by restricting the kinds of concept constructors allowed. The first, $\mathcal{FL}^-$ is obtained from $\mathcal{ALC}$ by restricting negation so that it can only appear in front of concept names. The second, $\mathcal{FL}_0$ is obtained by further restricting $\mathcal{FL}^-$ so that existential quantification only ranges over $\top$, i.e. $\exists R.\top$.

Both $\mathcal{FL}^-$ and $\mathcal{FL}_0$ can be thought of as being derived from the same based of constructors as $\mathcal{ALC}$. That is, both of these Description Logics contain the universal restriction concept constructor ($\forall$). This is mainly due to historical reasons. When researchers began to design DLs for knowledge representation they were influenced by early frame-based systems [Min75]. The universal restriction most closely corresponds to the intended semantics of slots and properties in the record like frame structures of frame based systems. However, during the last decade it was observed that many of the very large medical terminologies, such as SNOMED [SC97], the GENE ontology [ABB$^+$00], and GALEN [RNG93] which is written in the GRAIL Description Logic [RBG$^+$97], use existential ($\exists$) restrictions rather than universal restrictions. It was also observed that these styles of ontologies did not feature the kinds of non-determinism introduced by concept constructors such as concept-union ($\sqcup$). Out of these observations, the $\mathcal{EL}$ Description Logic [Bra04] was designed. $\mathcal{EL}$ features concept intersection and existential restrictions. This description logic is referred to as a *lightweight* DL as key reasoning tasks can be performed in polynomial time with the size of the input. Two extensions to $\mathcal{EL}$ are $\mathcal{EL}^+$ [BLS06b] and $\mathcal{EL}^{++}$ [BBL05], which increase the expressivity of $\mathcal{EL}$ while preserving the polynomial runtime behaviour of key reasoning tasks.

2.1.2 Highly Expressive Description Logics

By adding further modelling primitives (axiom types and complex concept constructors) to $\mathcal{ALC}$, Description Logics of higher expressivity are obtained, leading to the highly expressive Description Logic $\mathcal{SHOIN}$, which underpins OWL 1, the even more expressive Description Logic $\mathcal{SROIQ}$, which underpins OWL 2, and the Description Logic $\mathcal{SHOIQ}$, which falls between $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$. The names of Description Logics, which are drawn in a calligraphic font, give an idea as to the concept and role constructors that they support. For example,

- **Inverse Roles ($\mathcal{I}$)** — Inverse roles make it possible to refer to roles in both directions. For example, the $\mathsf{hasMother}^-$ refers to the *inverse* of $\mathsf{hasMother}$. If $\mathsf{hasMother}(\mathsf{Jean}, \mathsf{Maggie})$, then $\mathsf{hasMother}^-(\mathsf{Maggie}, \mathsf{Jean})$.

- **Role Hierarchy ($\mathcal{H}$)** — A role subsumption axiom specifies that one role is a sub-role of another role. For example, $\mathsf{hasSister} \sqsubseteq \mathsf{hasSibling}$ specifies that $\mathsf{hasSister}$ is a sub-role of $\mathsf{hasSibling}$. If $\mathsf{hasSister}(\mathsf{John}, \mathsf{Mary})$ then this implies $\mathsf{hasSibling}(\mathsf{John}, \mathsf{Mary})$.

- **Transitive Roles ($\mathcal{R}^+$)** — Transitive roles make it possible to describe the situation where a chain along a given role implies that role itself. For example, if hasAncestor is transitive, then if hasAncestor(Gemma, Jean) and hasAncestor(Jean, Maggie) then this implies hasAncestor(Gemma, Maggie).

- **Number Restrictions ($\mathcal{N}$)** — With number restrictions it is possible to count the number of role successors and predecessors for individuals. For example ≥ 4hasSibling is a concept that describes the individuals that have at least four hasSibling role successors. Similarly, ≤ 3hasSibling is a concept describes the individuals that have at most three hasSibling role successors.
- **Functional Roles ($\mathcal{F}$)** — A functional role states that for a given individual, a role can have no more than one value. For example, if an ABox contains the assertions hasMother(Jean, Peggy) and hasMother(Jean, Margaret), and the corresponding TBox contains $\top \sqsubseteq \leq 1$hasMother, which states that hasMother is functional, then it will be entailed that Peggy and Margaret are the same individual.
- **Nominals ($\mathcal{O}$)** — Nominals make it possible to construct singleton classes, which are classes that only have one individual as an instance. For example, {Airbus} is a class that only has one instance, namely Airbus. When combined with concept union, it is possible to use nominals to enumerate the members of a class. For example, { Airbus } $\sqcup$ { Boeing } is a concept with two instances and could be used to denote aircraft manufactures.
- **Qualified Number Restrictions ($\mathcal{Q}$)** — Qualified number restrictions are a more expressive form of number restrictions, which in contrast to plain number restrictions ($\mathcal{N}$) allow fillers of restrictions to be specified. For example ≥ 4 hasSibling.Brother specifies the class of individuals that have at least four hasSibling role successors to instances of Brother.

The letters which denote the various modelling primitives in a Description Logic are combined with each other to form the name of the Description Logic in question. For example, adding transitive roles to $\mathcal{ALC}$ gives $\mathcal{ALC}^{+}$, which is abbreviated to $\mathcal{S}$ [Sat96][1]. Adding role hierarchy and inverse roles to $\mathcal{S}$ gives $\mathcal{SHI}$ [HST00]. Adding number restrictions and nominals to $\mathcal{SHI}$ results in $\mathcal{SHOIN}$, which underpins OWL 1. An useful extension to $\mathcal{SHOIN}$ is obtained by replacing cardinality restrictions with qualified cardinality restrictions to give $\mathcal{SHOIQ}$ [HS07]. Finally adding complex role inclusion axioms, and some

[1]The abbreviation to $\mathcal{S}$ comes from the field of modal logic.

other features such as reflexive roles, to $\mathcal{SHOIQ}$ gives $\mathcal{SROIQ}$, which underpins OWL 2.

2.1.3 The Syntax and Semantics of $\mathcal{SHOIQ}$

This thesis focuses on the highly expressive Description Logic $\mathcal{SHOIQ}$, whose syntax and semantics are presented below.

$\mathcal{SHOIQ}$ Syntax

Let N_R be a set of role names. The set of $\mathcal{SHOIQ}$ roles is $N_R \cup \{R^- \mid R \in N_R\}$. For $\mathcal{SHOIQ}$ roles R and S, a $\mathcal{SHOIQ}$ role box consists of a finite set of role inclusion axioms of the form $R \sqsubseteq S$ and role equivalence axioms of the form $R \equiv S$ (where $R \equiv S$ is an abbreviation for $R \sqsubseteq S$ and $S \sqsubseteq R$), and a finite set of role assertions of the form $\mathsf{trans}(R)$, which asserts a role to be transitive. A role hierarchy is a finite set of role inclusion axioms.

Next, it is necessary to introduce the notion of a *simple* and *non-simple* roles. Again, full details may be found in [HS07], but intuitively, a role is non-simple if it is implied by a transitive role (via the reflexive transitive closure of the role hierarchy). For example, the axioms $\{\mathsf{trans}(R), R \sqsubseteq T, T \sqsubseteq P^-\}$ make the roles R, R^-, T, T^-, P^- and P non-simple. Roles are that are not non-simple are simple.

Let N_C be a set of concept names, and N_I be a set of individual names. The set of $\mathcal{SHOIQ}$ concepts is the smallest set such that

- every concept name $A \in N_C$, $\top$ and $\bot$ are concepts
- every nominal name $\{o\}$ for $o \in N_I$ is a concept
- if C and D are concepts, and R is a (possibly inverse) role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, are also concepts.
- if C is a concept, and R is a *simple* (possibly inverse) role, then $\geq nS.C$ and $\leq nS.C$ are also concepts.

A $\mathcal{SHOIQ}$ ontology is a tuple $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{T}$ is a TBox, $\mathcal{R}$ is an RBox and $\mathcal{A}$ is an ABox. A $\mathcal{SHOIQ}$ TBox is a finite set of general concept inclusion axioms (GCIs) of the form $C \sqsubseteq D$, and general concept equivalence axioms of the form $C \equiv D$ where C and D are $\mathcal{SHOIQ}$ concepts and $C \equiv D$ is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. A $\mathcal{SHOIQ}$ RBox is as described above. A $\mathcal{SHOIQ}$ ABox is a finite set of concept and role assertions of the form $C(a)$ and $R(a, b)$ respectively, where C is a $\mathcal{SHOIQ}$ concept, R is role name in N_R, and a and b are individuals in N_I.

Abbreviations

It should be noted that $\mathcal{SHOIQ}$ syntax contains several abbreviations, and OWL itself contains a significant amount of syntactic sugar, which are given below:

Constructor(s)	Abbreviation
$\mathsf{disjoint}(C, D)$	$C \sqsubseteq \neg D$
$\mathsf{domain}(R, C)$	$\exists R.\top \sqsubseteq C$
$\mathsf{range}(R, C)$	$\top \sqsubseteq \forall R.C$
$\mathsf{functional}(R)$	$\top \sqsubseteq \leq 1R$
$\mathsf{inversefunctional}(R)$	$\top \sqsubseteq \leq 1R^-$
$\mathsf{symmetric}(R)$	$R \equiv R^-$
$\mathsf{inverses}(R, S)$	$R \equiv S^-$
$= nR.C$	$\geq nR.C \sqcap \leq nR.C$
$\{o_1, \ldots, o_n\}$	$\{o_1\} \sqcup \cdots \sqcup \{o_n\}$

In addition, OWL permits the use of n-ary concept intersection, i.e. $C_1 \sqcap \cdots \sqcap C_n$ and n-ary concept union, i.e. $C_1 \sqcup \cdots \sqcup C_n$.

$\mathcal{SHOIQ}$ Semantics

The model-theoretic semantics of $\mathcal{SHOIQ}$ is specified using interpretations. An *interpretation* explicates the relationship between syntax and semantics, and is a binary tuple $\langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$, where $\Delta^\mathcal{I}$ is a non-empty set of objects called the *interpretation domain*, and $\cdot^\mathcal{I}$ is a function called the *interpretation function*. The interpretation function $\cdot^\mathcal{I}$ maps each concept name A into a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$, maps each role name R into a subset $R^\mathcal{I}$ of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and maps each individual name a into an element $a^\mathcal{I}$ of $\Delta^\mathcal{I}$.

Given an interpretation $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$, concepts C and D, and a role name R, the interpretation for $\mathcal{SHOIQ}$ concepts is defined defined as follows:

$$\begin{aligned}
\top^{\mathcal{I}} &:= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &:= \emptyset \\
(\neg C)^{\mathcal{I}} &:= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{x \mid \exists y.^{\mathcal{I}} \langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &:= \{x \mid \forall y.^{\mathcal{I}} \langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \\
(\geq nR.C)^{\mathcal{I}} &:= \{x \mid |\{y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \geq n\} \\
(\leq nR.C)^{\mathcal{I}} &:= \{x \mid |\{y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \leq n\}
\end{aligned}$$

The interpretation function for $\mathcal{SHOIQ}$ roles is defined as

$$(R^{-})^{\mathcal{I}} := \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\}$$

An interpretation $\mathcal{I}$ satisfies a $\mathcal{SHOIQ}$ axiom (written $\mathcal{I} \models \alpha$ for an arbitrary axiom α) as follows:

$\mathcal{I} \models C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

$\mathcal{I} \models R \sqsubseteq S$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

$\mathcal{I} \models \mathsf{trans}(R)$ if $\forall\, x, y, z,\ \langle x, y\rangle \in R^{\mathcal{I}}$ and $\langle y, z\rangle \in R^{\mathcal{I}}$ then $\langle x, z\rangle \in R^{\mathcal{I}}$

$\mathcal{I} \models C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$

$\mathcal{I} \models R(a, b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$

If an interpretation $\mathcal{I}$ satisfies an axiom α ($\mathcal{I} \models \alpha$) then $\mathcal{I}$ is known as a *model* of α ($\mathcal{I}$ is said to be a model α). An interpretation $\mathcal{I}$ satisfies (is a model of) an ontology, written $\mathcal{I} \models \mathcal{O}$, if $\mathcal{I}$ satisfies (is a model of) every axiom in $\mathcal{O}$.

Entailment An axiom α is *entailed* by an ontology $\mathcal{O}$, written $\mathcal{O} \models \alpha$ ($\mathcal{O}$ *entails* α), if $\mathcal{I} \models \alpha$ for every model $\mathcal{I}$ of $\mathcal{O}$. In particular, a concept C is *subsumed* by a concept D (D subsumes C), with respect to an ontology $\mathcal{O}$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$. An individual a is an *instance* of a class C, written $C(a)$, with respect to $\mathcal{O}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{O}$. A set of axioms $\mathcal{S}$ is entailed by an ontology $\mathcal{O}$, written $\mathcal{O} \models \mathcal{S}$ if $\mathcal{O} \models \alpha$ for every $\alpha \in \mathcal{S}$. An axiom α' is entailed by another axiom α, written $\alpha \models \alpha'$ if $\mathcal{I} \models \alpha'$ for every model $\mathcal{I}$ of α. An ontology does *not* entail an axiom α, written $\mathcal{O} \not\models \alpha$ if there exists a model $\mathcal{I}$ of $\mathcal{O}$ where $\mathcal{I} \not\models \alpha$. An ontology does *not* entail a set of axioms $\mathcal{S}$ if there is an $\alpha \in \mathcal{S}$ such that $\mathcal{O} \not\models \alpha$. An axiom α does *not* entail another axiom α' if there is a model $\mathcal{I}$ of α where $\mathcal{I} \not\models \alpha'$.

Satisfiability A concept C is *unsatisfiable* with respect to an ontology $\mathcal{O}$ if $C^{\mathcal{I}} = \emptyset$ in *every* model $\mathcal{I}$ of $\mathcal{O}$. In this case $\mathcal{O} \models C \sqsubseteq \bot$. Conversely, a concept C is *satisfiable* with respect to $\mathcal{O}$, if there *exists* a model $\mathcal{I}$ of $\mathcal{O}$ where $C^{\mathcal{I}} \neq \emptyset$.

Consistency An ontology $\mathcal{O}$ is *consistent* if there *exists* a model $\mathcal{I}$ of $\mathcal{O}$. An ontology $\mathcal{O}$ is *inconsistent* if there is no model $\mathcal{I}$ of $\mathcal{O}$ (there is no interpretation $\mathcal{I}$ of $\mathcal{O}$ that satisfies every axiom in $\mathcal{O}$). In this case, $\mathcal{O} \models \top \sqsubseteq \bot$. An inconsistent ontology entails everything. The effect of this is that, using standard classic semantics, no meaningful conclusions can be drawn from an inconsistent ontology.

Axiom Strength and Deductive Closures

Using the notion of entailment, it is possible to talk about *axiom strength* and the *deductive closure* of an ontology. It should be noted that these definitions are quite general and are not specific to $\mathcal{SHOIQ}$.

Axiom Strength An axiom α' is said to be *weaker* than another axiom, α if $\alpha \models \alpha'$ but $\alpha' \not\models \alpha$. In this case, α is also said to be *stronger* than α'. For example, $A \sqsubseteq \exists R.C \models A \sqsubseteq \exists R.\top$, but $A \sqsubseteq \exists R.\top \not\models A \sqsubseteq \exists R.C$, and so $A \sqsubseteq \exists R.\top$ is weaker than $A \sqsubseteq \exists R.C$. The same notion is applicable to sets of axioms. A set of axioms $\mathcal{S}'$ is weaker than a set of axioms $\mathcal{S}$ if $\mathcal{S} \models \mathcal{S}'$, but $\mathcal{S}' \not\models \mathcal{S}$.

Deductive Closures Given an ontology, $\mathcal{O}$, and a description logic $\mathcal{L}$ the *deductive closure* of $\mathcal{O}$, is written as $\mathcal{O}^{\star}_{\mathcal{L}}$, where $\mathcal{O}^{\star}_{\mathcal{L}} = \{\alpha \in \mathcal{L} \mid \mathcal{O} \models \alpha\}$. In other words the deductive closure contains *all* well formed $\mathcal{L}$-axioms that are entailed by the ontology $\mathcal{O}$. When it is clear from the context, the subscript $\mathcal{L}$ is usually dropped.

2.1.4 Key Reasoning Tasks

In the context of Description Logics, reasoning is the process used to determine the entailments that follow from an ontology. In the DL world there are several, so called, "standard" reasoning tasks which are described below.

Consistency Checking Given an ontology $\mathcal{O}$ as an input, is $\mathcal{O}$ consistent? i.e. is there at least one model $\mathcal{I}$ of $\mathcal{O}$.

Satisfiability Checking Given an ontology $\mathcal{O}$ and a class expression C, is C satisfiable with respect to $\mathcal{O}$ ($\mathcal{O} \not\models C \sqsubseteq \bot$)? i.e, is it the case that $C^{\mathcal{I}} \neq \emptyset$ in *some* model $\mathcal{I}$ of $\mathcal{O}$?

Subsumption Testing Given an ontology $\mathcal{O}$ and two class expressions C and D, is it the case that $C \sqsubseteq D$ with respect to $\mathcal{O}$ ($\mathcal{O} \models C \sqsubseteq D$)? i.e., is it the case that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{O}$?

Instance Checking Given an ontology $\mathcal{O}$, a class expression C and an individual a, is it the case that a is an instance of C with respect $\mathcal{O}$ ($\mathcal{O} \models C(a)$). i.e., is the case that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{O}$?

Reduction to Consistency Checking Consistency checking can be regarded as the main standard reasoning service. In practice, for logics such as $\mathcal{SHOIQ}$, where it is possible to have inconsistent ontologies, consistency checking is performed before any other reasoning. Moreover, for $\mathcal{SHOIQ}$ ontologies, the three other standard reasoning services are reducible to consistency checking: A concept C is unsatisfiable with respect to $\mathcal{O}$ if and only if $\mathcal{O} \cup \{C(x)\}$ is inconsistent for some fresh individual name x; A concept C is subsumed by D ($C \sqsubseteq D$) with respect to $\mathcal{O}$ if and only if $\mathcal{O} \cup \{(C \sqcap \neg D)(x)\}$ is inconsistent for some fresh individual name x, and; An individual a is an instance of a concept C if and only if $\mathcal{O} \cup \{\neg C(a)\}$ is inconsistent.

2.1.5 Structural Notions

This thesis uses several standard syntactic and structural notions which are defined below.

Signature The signature of a (complex) concept, axiom, or set of axioms is the set of concept, role, and individual names appearing in the (complex) concept, axiom, or set of axioms. $\mathsf{signature}(X)$ denotes the signature of X where X is either a concept, axiom, or set of axioms. For example, consider the ontology $\mathcal{O} = \{A \sqsubseteq \exists R^{-}.C, B \sqsubseteq \forall S.D\}$. The signature of $\mathcal{O}$ ($\mathsf{signature}(\mathcal{O})$) is $\{A, R, C, B, S, D\}$. It should be noted that the signature of a concept, axiom or set of axioms does not contain the concepts $\top$ or $\bot$.

Subconcepts A concept C is a *subconcept* of D if C *syntactically* occurs in D. For example, the concept $\neg(A \sqcap \exists R.C)$, contains the subconcepts $A \sqcap \exists R.C$, A, $\exists R.C$ and C.

Position A *position* is a finite sequence of integers written as $i_1.i_2.\ldots.i_n$. The empty position is written as ϵ. The position p of a concept, role, or individual in an axiom α is written as $\alpha|_p$. Similarly, the position p of a subconcept, role, or individual in a concept C is written as $C|p$. The positions of subconcepts, roles and individuals in $\mathcal{SHOIQ}$ axioms or concepts is recursively defined in Table 2.2. For an example of positions involving complex concepts consider

$$\alpha = A \sqsubseteq \exists R.(C \sqcap D)$$

which contains the following positions $\alpha|_1 = A$, $\alpha|_2 = \exists R.(C \sqcap D)$, $\alpha|_{2.1} = R$, $\alpha|_{2.2} = C \sqcap D$, $\alpha|_{2.2.1} = C$ and $\alpha|_{2.2.2} = D$. A substitution of one subconcept at a position p with another concept C in an axiom α is written as $\alpha[C]_p$. Taking the example above, $\alpha[\top]_{2.2} = A \sqsubseteq \exists R.\top$.

Polarity The *polarity* [Mur82] of a concept C at position p in an axiom or concept is defined recursively in Table 2.2. Notice how the polarity of nested subconcepts within negation and maximum cardinality restrictions "flips" from the polarity immediately outside the negation or filler. For an example of polarity consider

$$\alpha = A \sqsubseteq B \sqcap \neg(\exists R.C \sqcup \leq nR.\neg D)$$

Table 2.2: Sub-Term Position and Polarity

Term	Term Label	Positions	Polarity
$C \sqsubseteq D$	α	$\alpha\|_1 = C$ $\alpha\|_2 = D$	$\mathsf{polarity}(\alpha\|_1) = -$ $\mathsf{polarity}(\alpha\|_2) = +$
$C(a)$	α	$\alpha\|_1 = C$ $\alpha\|_2 = a$	$\mathsf{polarity}(\alpha\|_1) = +$
$D \sqcap E$ or $D \sqcup E$	C	$C\|_1 = D$ $C\|_2 = E$	$\mathsf{polarity}(C_1) = \mathsf{polarity}(C)$ $\mathsf{polarity}(C_2) = \mathsf{polarity}(C)$
$\neg D$	C	$C\|_1 = D$	$\mathsf{polarity}(C_1) = -\mathsf{polarity}(C)$
$\exists R.D$ or $\forall R.D$	C	$C\|_1 = R$ $C\|_2 = D$	 $\mathsf{polarity}(C\|_2) = \mathsf{polarity}(C)$
$\geq nR.D$	C	$C\|_1 = n$ $C\|_2 = R$ $C\|_3 = D$	 $\mathsf{polarity}(C\|_3) = \mathsf{polarity}(C)$
$\leq nR.D$	C	$C\|_1 = n$ $C\|_2 = R$ $C\|_3 = D$	 $\mathsf{polarity}(C\|_3) = -\mathsf{polarity}(C)$
$\{o\}$	C	$C\|_1 = o$	

the polarity of some subconcept occurrences is

$$\begin{aligned}
\mathsf{polarity}(\alpha|_1 = A) &= - \\
\mathsf{polarity}(\alpha|_{2.1} = B) &= + \\
\mathsf{polarity}(\alpha|_{2.2.1} = (\exists R.C \sqcup \leq nR.\neg D)) &= - \\
\mathsf{polarity}(\alpha|_{2.2.1.2.3} = \neg D) &= +
\end{aligned}$$

Structural Equality For any concepts C and D the concept $C \sqcap D$ is structurally equivalent to the concept $D \sqcap C$. In other words, the order of operands is not important when it comes to structural equality. Similarly, the concept $C \sqcup D$ is structurally equivalent to the concept $D \sqcup C$. This means that $A \sqsubseteq B \sqcap \exists R.C$ is considered to be a repetition of $A \sqsubseteq \exists R.C \sqcap B$ and vice-versa. This notion of structural equality is borrowed from the OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax [MPSP09].

2.2 Reasoning

As mentioned previously, reasoning is the procedure used to determine whether or not an axiom is entailed by a set of axioms. Entailment checking in $\mathcal{SHOIQ}$ (which can be reduced to ontology consistency checking) is decidable [HS07], which means that an entailment checking procedure will *terminate* after a *finite* amount of time with a "yes" or "no" answer.

Tableau Algorithms Tableau algorithms [BS01] are a common kind of algorithm for reasoning in the area of Description Logics. Implementations of them are found in many highly optimised OWL reasoners such as FaCT++ [TH06], Pellet [SPG$^+$07] and Racer [HM01]. In order to determine whether an input ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R} \rangle$ is consistent or not a tableau algorithm attempts to build a (representation of a) model of $\mathcal{O}$. If the algorithm succeeds in building a model of $\mathcal{O}$ then it answers "yes" (consistent). If the algorithm fails to build a model of $\mathcal{O}$ then it answers "no" (inconsistent). Since subsumption checking and satisfiability checking can be reduced to consistency checking, to check whether $\mathcal{O} \models C \sqsubseteq D$, a tableau reasoner will check whether $\mathcal{O}' = \mathcal{O} \cup \{(C \sqcap \neg D)(x_0)\}$, where x_0 is a fresh individual name, is consistent. If the reasoner finds $\mathcal{O}'$ to be inconsistent, then $\mathcal{O} \models C \sqsubseteq D$.

Given an input ontology $\mathcal{O}$, the basic idea behind a tableau algorithm is to produce a finite representation of a model (producing a series of partial representations of the model along the way). The representation is a set of ABoxes, written as $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, where the assertions in each ABox represent constraints on the associated model. For example, if a must be an instance of C in the model then the representative ABox contains a concept assertion $C(a)$. In $\mathcal{ALC}$, and other languages that have non-determinism built in, a *set* of ABoxes is generated by a tableau algorithm because each ABox represents a *choice point* in the construction of a model. For example, given $(C \sqcup D)(a)$, one class of models exists where a is an instance of C, thus one ABox would contain the assertion $C(a)$, and another ABox would contain the assertion $D(a)$ to represent a being an instance of D in the other class of models. Ultimately, each generated ABox

Table 2.3: Tableau Rules for $\mathcal{ALC}$ Concept Satisfiability

Rule	**Condition and Action**
$\sqcap$**-rule**	If $(C \sqcap D)(x) \in \mathcal{A}$ but $C(x) \notin \mathcal{A}$ *and* $C(x) \notin \mathcal{A}$ then $\mathcal{A}' \leftarrow \mathcal{A} \cup \{C(x), D(x)\}$
$\sqcup$**-rule**	If $(C \sqcup D)(x) \in \mathcal{A}$ but $C(x) \notin \mathcal{A}$ or $C(x) \notin \mathcal{A}$ then $\mathcal{A}' \leftarrow \mathcal{A} \cup \{C(x)\}$ and $\mathcal{A}'' \leftarrow A \cup \{D(x)\}$
$\exists$**-rule**	If $(\exists R.C)(x) \in \mathcal{A}$ but $R(x,y) \notin \mathcal{A}$ *and* $C(y) \notin \mathcal{A}$ for some y then $\mathcal{A}' \leftarrow \mathcal{A} \cup \{R(x,y), C(y)\}$
$\forall$**-rule**	If $\forall R.C(x) \in \mathcal{A}$ and $R(x,y) \in \mathcal{A}$ but $C(y) \notin \mathcal{A}$ then then $\mathcal{A}' \leftarrow \mathcal{A} \cup \{C(y)\}$

contains assertions that are derived by the recursive and exhaustive application of a set of *tableau rules*. For any given language $\mathcal{L}$, such as $\mathcal{SHOIQ}$, there is a set of *tableau rules* that is defined. Generally, there is one tableau rule per class constructor, with each rule reflecting the semantics of the constructor. Notice how each rule consists of a precondition and an action, where the action is only applied if each precondition is met. The tableau rules for $\mathcal{ALC}$ [SSS91] are shown in Table 2.3. For a set of ABoxes $\mathcal{S}$, the application of a rule on an ABox $\mathcal{A}$ results in the replacement of $\mathcal{A}$ with a new ABox $\mathcal{A}'$, or the the replacement of $\mathcal{A}$ with two new ABoxes $\mathcal{A}'$ and $\mathcal{A}''$. In both cases, $\mathcal{A}'$ and $\mathcal{A}''$ are supersets of $\mathcal{A}$.

A Tableau Algorithm for $\mathcal{ALC}$ Concept Satisfiability In order to illustrate the main ideas behind tableau algorithms, a tableau algorithm for $\mathcal{ALC}$ *concept satisfiability*, i.e. satisfiability of concepts with respect to an empty TBox, is presented below. Satisfiability with respect to a TBox is briefly discussed later.

In order to test the satisfiability of a concept C the $\mathcal{ALC}$ concept satisfiability tableau algorithm begins by initialising a set of ABoxes $\mathcal{S}$ with a single ABox $\mathcal{A}$ that contains a single assertion $C(x)$. Next, the algorithm proceeds to expand $\mathcal{A}$ and $\mathcal{S}$ by exhaustive application of the tableau rules shown in Table 2.3. The tableau algorithm terminates when *no more rules can be applied*, or when *all* ABoxes $\{A_1, \ldots, \mathcal{A}_n\} \subseteq \mathcal{S}$ contain a *clash*. For $\mathcal{ALC}$, a clash is of the form $\mathcal{A}_i = \{\ldots, A(x), \neg A(x), \ldots\}$ or $\{\ldots, \bot(x)\}$ for some individual x. Every ABox $\mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ contains at least one clash if and only if $\mathcal{A}$ is inconsistent.

As an example consider the concept $(\exists R.B \sqcap \forall R.(\neg B \sqcup E))$, which is satisfiable.

The algorithm first initialises the set of ABoxes to

$$\mathcal{S} = \{ \;\; \mathcal{A}_1 = \{(\exists R.B \sqcap \forall R.(\neg B \sqcup E))(x)\} \;\; \}$$

Next, it applies the $\sqcap$-rule, since the initialised ABox contains a top-level conjunction. This gives

$$\begin{aligned}\mathcal{S} = \{ \;\; \mathcal{A}_1 = \{ \; & (\exists R.B \sqcap \forall R.(\neg B \sqcup E))(x) \\ & (\exists R.B)(x) \\ & (\forall R.(\neg B \sqcup E))(x) \; \} \;\; \}\end{aligned}$$

Next, the $\exists$-rule is applied and $\mathcal{A}_1$ is extended so that

$$\begin{aligned}\mathcal{S} = \{ \;\; \mathcal{A}_1 = \{ \; & (\exists R.B \sqcap \forall R.(\neg B \sqcup E))(x) \\ & (\exists R.B)(x) \\ & (\forall R.(\neg B \sqcup E))(x) \\ & R(x, y) \\ & B(y) \; \} \;\; \}\end{aligned}$$

The $\forall$-rule is then applied to give

$$\begin{aligned}\mathcal{S} = \{ \;\; \mathcal{A}_1 = \{ \; & (\exists R.B \sqcap \forall R.(\neg B \sqcup E))(x) \\ & (\exists R.B)(x) \\ & (\forall R.(\neg B \sqcup E))(x) \\ & R(x, y) \\ & B(y) \\ & (\neg B \sqcup E)(y) \; \} \;\; \}\end{aligned}$$

Finally, the $\sqcup$-rule is applied. The non-determinism, or choice, aspect of this rule causes $\mathcal{A}_1$ to be replaced with *two* ABoxes $\mathcal{A}_2$ and $\mathcal{A}_3$. Each reflecting the choice in that y can be an instance of $\neg B$ or it can be an instance of E.

$$
\begin{array}{lll}
\mathcal{S} = \{ & \mathcal{A}_2 = \{ & (\exists R.B \sqcap \forall R.(\neg B \sqcup E))(x) \\
 & & (\exists R.B)(x) \\
 & & (\forall R.(\neg B \sqcup E))(x) \\
 & & R(x,y) \\
 & & B(y) \\
 & & (\neg B \sqcup E)(y) \\
 & & (\neg B)(y) \ \} \\
 & & \\
 & \mathcal{A}_3 = \{ & (\exists R.B \sqcap \forall R.(\neg B \sqcup E))(x) \\
 & & (\exists R.B)(x) \\
 & & (\forall R.(\neg B \sqcup E))(x) \\
 & & R(x,y) \\
 & & B(y) \\
 & & (\neg B \sqcup E)(y) \ \} \\
 & & E(y)\} \ \ \}
\end{array}
$$

At this stage no more rules can be applied to any of the ABoxes in $\mathcal{S}$. The algorithm checks the ABoxes to see if there is at least one of them that does not contain a clash. In this case, the ABox $\mathcal{A}_2$ does contain a clash, but the ABox $\mathcal{A}_3$ does *not* contain a clash. Therefore there is complete and clash free ABox and the algorithm returns "true" i.e. it is possible for there to be a model of the input concept where it is not interpreted as the empty set, so the input concept is satisfiable.

Dealing with TBox Axioms The above tableau algorithm for $\mathcal{ALC}$ concept satisfiability can be extended into an algorithm for testing satisfiability with respect to a TBox $\mathcal{T}$ by adding extra tableau rules and, for *cyclic* TBoxes, blocking conditions. In order to deal with concept inclusion axioms in a TBox, the two rules shown in Table 2.4 need to be added. The first rule ($\sqsubseteq_1$-rule) is an "unfolding" rule that performs lazy unfolding [Hor97] for concept inclusion axioms of the form $A \sqsubseteq C$ (i.e. where A is a concept name). The second rule ($\sqsubseteq_2$-rule) deals with general concept inclusion axioms of the form $C \sqsubseteq D$, where the subclass is a complex concept. It is not possible to perform unfolding on complex concepts and so this rule adds the assertion $(\neg C \sqcup D)(x)$ for every individual x in an ABox (which reflects the semantics of subsumption). In the presence of a TBox that contains general concept inclusion axioms or *cycles* the tableau algorithm, as it is, may not terminate. To see why this is the case, consider the cyclic TBox $\mathcal{T} = \{A \sqsubseteq \exists R.A\}$ (the TBox $\{\top \sqsubseteq \exists R.A$ would also suffice). To test the satisfiability of A, the algorithm begins with

$$\mathcal{S} = \{\ \ \mathcal{A} = \{(A)(x)\}\ \ \}$$

Next, it applies the unfolding rule ($\sqsubseteq_1$-rule) to $A(x)$ to give

$$\begin{aligned}\mathcal{S} = \{\ \ \mathcal{A} = \{&(A)(x)\\ &(\exists R.A)(x)\}\ \ \}\end{aligned}$$

This is followed by the $\exists$-rule to give

$$\begin{aligned}\mathcal{S} = \{\ \ \mathcal{A} = \{&(A)(x)\\ &(\exists R.A)(x)\\ &R(x, y)\\ &A(y)\}\ \ \}\end{aligned}$$

Notice that a fresh individual y has been generated, which is also an instance of A. The three steps above get repeated for $A(y)$, which generates another fresh individual that is an instance of A and the process repeats for ever.

To get around the non-termination problem it is necessary to introduce the notion of *blocking*. Intuitively, individuals in an ABox can become *blocked* so that generating tableau rules (rules which introduce fresh individuals into an ABox) do not apply to them. Restricting rule application in this way means that cycles like the one above do not cause infinite rule application, and the termination property of the algorithm is regained. In $\mathcal{ALC}$, a form of blocking known as *subset blocking* is used. First is necessary to introduce the notion of a blocked individual: An individual y is blocked by an individual x (where $x \neq y$) in an ABox $\mathcal{A}$ if $\{D \mid D(y) \in \mathcal{A}\} \subseteq \{D \mid D(x) \in \mathcal{A}\}$. Next, the generating $\exists$-rule is modified to ensure that the rule is not applicable to an individual x if it is a

Table 2.4: Unfolding Rules for $\mathcal{ALC}$ Concept Satisfiability

Rule	**Condition and Action**
$\sqsubseteq_1$**-rule**	If $A(x) \in \mathcal{A}$ and $A \sqsubseteq C \in \mathcal{T}$ and $C(x) \notin \mathcal{A}$ then $\mathcal{A}' \leftarrow \mathcal{A} \cup \{C(x)\}$
$\sqsubseteq_2$**-rule**	If $C \sqsubseteq D \in \mathcal{T}$ and $(\neg C \sqcup D)(x) \notin \mathcal{A}$ then $\mathcal{A}' \leftarrow \mathcal{A} \cup \{(\neg C \sqcup D)(x)\}$

blocked individual.

Tableau Algorithms for More Expressive Logics Tableau algorithms for more expressive logics can be obtained by adding extra tableau rules and replacing simple subset blocking with more elaborate blocking strategies.

2.3 Justifications

A justification [Kal06, BH95, SC03] for an entailment in an ontology is a *minimal* subset of the ontology that is sufficient for the entailment to hold. The justification is a minimal subset in that the entailment in question does not follow from any proper subset of the justification. More precisely,

Definition 1 (Justification)**.** *$\mathcal{J}$ is a justification for $\mathcal{O} \models \eta$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subsetneq \mathcal{J}$ it is the case that $\mathcal{J}' \not\models \eta$.*

Justifications and their variants have also been known by other names. In particular, MUPS [SC03] and MinAs [Sun09].

2.3.1 The Number of Justifications for an Entailment

In work by Baader et al. [BPS07] it was shown that, even for very inexpressive ontology languages, the number of justifications for an entailment in an ontology can be exponential in the size of the ontology. Consider the following ontology

$$\mathcal{O} = \{A_{i-1} \sqsubseteq B_i \sqcap C_i, \quad B_i \sqsubseteq A_i, \quad C_i \sqsubseteq A_i \mid 1 \leq i \leq n\}$$

For all $n \geq 1$, the size of $\mathcal{O}$ is linear in n. Now consider the justifications for $\mathcal{O} \models A_0 \sqsubseteq A_n$. There are 2^n justifications for this entailment with respect to $\mathcal{O}$.

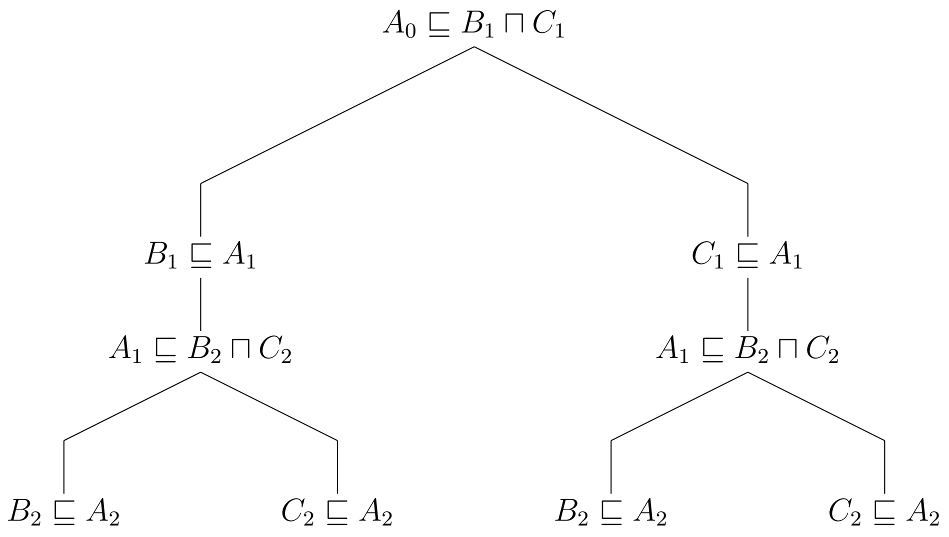


Figure 2.1: The number of justifications can be exponential in the size of the ontology. Each path through the tree above represents a justification for the entailment $A_0 \sqsubseteq A_2$ in the ontology $\mathcal{O} = \{A_{i-1} \sqsubseteq B_i \sqcap C_i,\ \ B_i \sqsubseteq A_i,\ \ C_i \sqsubseteq A_i \mid 1 \leq i \leq n\}$ for $n = 2$. $\mathcal{O}$ grows linearly in the size of n, but the number of justifications for $A_{i-1} \sqsubseteq A_n$ is 2^n

This is due to the fact that, as depicted in Figure 2.1, for each $i(1 \leq i \leq n)$ there is a binary choice of which axiom to include in a justification.

The above example uses explicit conjunction to illustrate how there can be an exponential number of justifications for an entailment in an ontology. However, the example works just as well without explicit conjunctions:

$$\mathcal{O} = \{A_{i-1} \sqsubseteq B_i, A_{i-1} \sqsubseteq C_i,\ \ B_i \sqsubseteq A_i,\ \ C_i \sqsubseteq A_i \mid 1 \leq i \leq n\}$$

This illustrates the point that there can be an exponential number of justifications in an ontology containing only subsumption between concept names that is built from a weak ontology language such as RDF Schema (RDFS) [BG09].

2.3.2 Root and Derived Unsatisfiable Classes

In his thesis [Kal06], Kalyanpur introduced the notion of *root unsatisfiable classes* and *derived unsatisfiable classes*. Intuitively, given an ontology $\mathcal{O}$, whose signature contains unsatisfiable concept names, a *root unsatisfiable class* is a concept name in the signature of $\mathcal{O}$ whose unsatisfiability does not depend on the unsatisfiability of any other concept name in the signature of $\mathcal{O}$. A *derived unsatisfiable* *class* is a concept name in the signature of $\mathcal{O}$ whose unsatisfiability depends on the unsatisfiability of some other concept in the signature of $\mathcal{O}$. More precisely,

Definition 2 (Root/Derived Unsatisfiable Classes)**.** *Given an ontology $\mathcal{O}$ where $\mathcal{O} \models A \sqsubseteq \bot$, A is a* ***derived unsatisfiable class*** *if there exists some other class B in the signature of $\mathcal{O}$, where $A \neq B$, such that $\mathcal{O} \models B \sqsubseteq \bot$ and there is a justification $\mathcal{J}_A$ with respect to $\mathcal{O}$ for $\mathcal{O} \models A \sqsubseteq \bot$ and another justification $\mathcal{J}_B$ with respect to $\mathcal{O}$ for $\mathcal{O} \models B \sqsubseteq \bot$ such that $\mathcal{J}_A \subsetneq \mathcal{J}_B$. Any unsatisfiable concept name in the signature of $\mathcal{O}$ which is not a derived unsatisfiable class is a* ***root unsatisfiable class****.*

2.3.3 Justification Based Repair

Given the set of all justifications $\{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$ for an entailment η in an ontology $\mathcal{O}$ it is possible to to use them to *break* η and *repair* $\mathcal{O}$ so that $\mathcal{O} \not\models \eta$. This is done by choosing one axiom from each justification $\mathcal{J}_i \in \{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$ $(1 \leq i \leq n)$ and removing these axioms from $\mathcal{O}$. Since each justification is a minimally entailing set of axioms, removing these axioms from $\mathcal{O}$ must break the entailment. The set of axioms $\mathcal{R}$ which is made up of *one* axiom from each justification is known as a *repair*. Definition 3 makes the notion of a repair more precise.

Definition 3 (Repair)**.** *Given $\mathcal{O} \models \eta$, the set of axioms $\mathcal{R}$ is a repair for η in $\mathcal{O}$ if $\mathcal{R} \subseteq \mathcal{O}$, $\mathcal{O} \setminus \mathcal{R} \not\models \eta$ and there is no $\mathcal{R}' \subsetneq \mathcal{R}$ such that $\mathcal{O} \setminus \mathcal{R}' \not\models \eta$.*

The *smallest* set of axioms that is a repair is known as a cardinality minimal repair. More precisely,

Definition 4 (Cardinality Minimal Repair)**.** *Given for $\mathcal{O} \models \eta$, the set of axioms $\mathcal{R}$ is a cardinality minimal repair if $\mathcal{R}$ is a repair for η in $\mathcal{O}$ and there is no other repair $\mathcal{R}'$ for η in $\mathcal{O}$ such that $|\mathcal{R}'| < |\mathcal{R}|$.*

2.4 Ontology Modularisation

In recent years, there has been a significant amount of research effort spent on studying ontology modularisation. Various notions of modularity have been defined, and the properties of the modules that they give rise to have been investigated in depth. In the most general sense, a module $\mathcal{M}$ of an ontology $\mathcal{O}$ is simply a well defined subset of $\mathcal{O}$ that has some desirable properties. Roughly speaking, there are two broad notions of modularity, *syntactic based modularity* and *semantic based modularity.* Syntactic based approaches such as the one used in the Protégé PROMPT tool suite [NM03], or the one used by Rector and Seidenberg to extract modules from SNOMED [SR06], only take into consideration the shared signature between axioms in an ontology when extracting a module from that ontology—they do not pay attention to the semantics of the language. In general, their goal is to extract a small portion of an ontology that is useful for some purpose. Purely syntactic approaches make no guarantee of producing modules that preserve certain classes of entailments from original ontology. In contrast, semantic based approaches [CHKS07, Sun09] pay attention to the semantics of the language, and produce modules that preserve classes of entailments that hold in the original ontology. For example, a *subsumption-module* [Sun09] $\mathcal{M}$ for an ontology $\mathcal{O}$ and a concept name A is a subset of $\mathcal{O}$ which is guaranteed to preserve all atomic subsumptions of the form $A \sqsubseteq B$ that hold with respect to $\mathcal{O}$ for any concept name B in the signature of $\mathcal{O}$. Similarly, a locality based module [CHKS07] $\mathcal{M} \subseteq \mathcal{O}$ for some ontology $\mathcal{O}$ and signature $\Sigma \subseteq \mathsf{signature}(\mathcal{O})$ is guaranteed to preserve all entailments α that can be built from entities in Σ which hold in $\mathcal{O}$.

2.4.1 Modularisation and Justifications

With regards to justification finding, the basic idea is that rather than computing justifications with respect to some ontology $\mathcal{O}$, justifications are computed with respect to a module of $\mathcal{O}$ for some entailment η (or signature of η). The hope is that $\mathcal{M}$ is "much" smaller than $\mathcal{O}$, that it can be efficiently computed (in polynomial time in the size of $\mathcal{O}$) meaning it provides a smaller search space in comparison to $\mathcal{O}$, and that it also offers the possibility of a boost in entailment checking performance.

Obviously, given $\mathcal{O} \models \eta$ and a module $\mathcal{M} \subseteq \mathcal{O}$ for η (or the signature of η), which is guaranteed to preserve η, i.e. $\mathcal{M} \models \eta$, there are two possibilities: (1) $\mathcal{M}$ contains *at least one* justification for η with respect to $\mathcal{O}$; or (2) $\mathcal{M}$ contains *all* justifications for η with respect to $\mathcal{O}$. Most research has focused on the later possibility, and their are various types of modules for various languages that are guaranteed to contain all justifications for an entailment. For example, *strong subsumption modules* [BS08] and their various implementations (e.g. *reachability based modules* [Sun09], and *bi-directional reachability-based modules* [NBM09]), or *syntactic locality based modules* [CHKS07], which were shown to be *depleting* [SSZ09], meaning that they contain all justifications for a given entailment.

2.4.2 Syntactic Locality Based Modules

In this thesis *syntactic locality based modules* [CHKS07], which are based on the notion of conservative extensions [GLW06], are used as an optimisation for computing justifications. Given a $\mathcal{SHOIQ}$ ontology O, where $\mathcal{O} \models \eta$, a syntactic locality based module for the signature of η can be computed in polynomial time, results in all entailments that can be built from the signature η (obviously including η itself), and contains all justifications for η.

Syntactic locality based modules for $\mathcal{SHOIQ}$ are defined as follows, where it is assumed that $C \sqcup D$ is an abbreviation for $\neg(\neg C \sqcap \neg D)$, $\forall R.C$ an abbreviation for $\neg(\exists R.\neg C)$ and $\leq nR.C$ an abbreviation for $\neg(\geq (n+1)R.C)$: Let Σ be a signature. The following grammar recursively defines two sets of concepts $\mathcal{C}^{\perp}_{\Sigma}$ and $\mathcal{C}^{\top}_{\Sigma}$ for a signature Σ:

$$\begin{aligned}
\mathcal{C}^{\perp}_{\Sigma} &::= A^{\perp} \mid (\neg C^{\top}) \mid (C \sqcap C^{\perp}) \mid (\exists R^{\perp}.C) \mid (\exists R.C^{\perp}) \mid (\geq nR^{\perp}.C) \mid (\geq nR.C^{\perp}) \\
\mathcal{C}^{\top}_{\Sigma} &::= (\neg C^{\perp}) \mid (C^{\top}_{1} \sqcap C^{\top}_{2})
\end{aligned}$$

where $A^{\perp} \notin \Sigma$ is a concept name, R is a role, C is a concept, $C^{\perp} \in \mathcal{C}^{\perp}_{\Sigma}$, $C^{\top}_{(i)} \in \mathcal{C}^{\top}_{\Sigma}$ for $i = 1, 2$, and $R^{\perp} \notin \Sigma$ is a role. An axiom is *syntactically local* with respect to Σ if it is one of the following forms: (1) $R^{\perp} \sqsubseteq R$, or (2) $\mathsf{trans}(R)$, or (3) $C^{\perp} \sqsubseteq C$, or (4) $C \sqsubseteq C^{\top}$. The set of all axioms that are syntactically local with respect to Σ is denoted by $\mathsf{s\_local}(\Sigma)$. A $\mathcal{SHOIQ}$ ontology $\mathcal{O}$ is syntactically local with respect to Σ if $\mathcal{O} \subseteq \mathsf{s\_local}(\Sigma)$.

Intuitively, every concept in $\mathcal{C}^{\perp}_{\Sigma}$ becomes equivalent to $\perp$, and every concept in $\mathcal{C}^{\top}_{\Sigma}$ becomes equivalent to $\top$, if every concept name $A^{\perp} \notin \Sigma$ is replaced with $\perp$, and every role name $R^{\perp} \notin \Sigma$ is replaced with the empty role. This means that syntactically local axioms become tautologies after these replacements.

Syntactic locality based modules are then defined as follows: Let $\mathcal{O}$ be a $\mathcal{SHOIQ}$ ontology, let $\mathcal{O}' \subseteq \mathcal{O}$, and Σ a signature. Then $\mathcal{O}'$ is a syntactic locality based module for Σ if every axiom $\alpha \in \mathcal{O} \setminus \mathcal{O}'$ is syntactically local with respect to $\Sigma \cup \mathsf{signature}(\mathcal{O}')$.

In work by Grau et al. [GHKS08] it was shown that there is always a unique minimal module for an ontology $\mathcal{O}$ and signature Σ. More details of how modularisation is used as an optimisation in justification finding are given in the next chapter.

Chapter 3

Computing Justifications

This chapter focuses on computing justifications for entailments in ontologies. An implementation of a justification finding service is a key component in many of the explanation and debugging tools that exist for ontology development environments such as Swoop [KPH05, Kal06], the RaDON plugin for the NeOn Toolkit [JHQ+09], the explanation workbench for Protégé-4 [HPS08a], the explanation facility in OWL Sight [Gro09], and the explanation view in TopBraid Composer [Knu07]. Justification finding services are also increasingly being used as auxiliary services in other applications for example in incremental reasoning [CHWK07], reasoning over very large ABoxes [DFK+07], belief base revision [HWKP06], meta-modelling support [GRV10], default reasoning [SdF+10] and eliminating redundant axioms in ontologies [GW11]. In later chapters, black-box algorithms for computing *laconic justifications* (Chapter 10) and *Justification Oriented Proofs* (13) are presented, and these algorithms require subroutines for computing justifications—hence they use justification finding services as auxiliary services.

The overall aim of this chapter is to present a thorough investigation into the practicalities of computing all justifications for entailments in published ontologies. In particular, ontologies which are representative of typical modelling and are not tutorial or reasoner test-bed ontologies. This chapter also investigates how *black-box* justification finding stacks up against *glass-box* justification finding[1], investigates the effect of various optimisations, and provides a detailed picture on the "justification landscape" for real world ontologies.

[1]See Section 3.1 below for definitions of these terms.

3.1 Techniques for Computing Justifications

In general, algorithms for computing justifications are described using two axes of classification. The first, the *single-all-axis* is whether an algorithm computes a *single* justification for an entailment or whether it computes *all* justifications for an entailment. The second, the *reasoner-coupling-axis* is whether the algorithm is a *black-box* algorithm or whether it is a *glass-box* algorithm.

The Single-All Axis: One Justification versus All Justifications Algorithms In practice it is useful to distinguish between algorithms that compute a *single* justification for an entailment, and algorithms that compute *all* justifications for an entailment. There are two basic reasons for this: (1) Algorithms for computing all justifications tend to depend on algorithms for computing single justifications as sub-routines. Indeed, in the empirical investigation that follows, three single justification finding algorithm variants are used as sub-routines by algorithm for computing all justifications. (2) In application scenarios, where justifications are used as explanations for human users, rather than being used for the purposes of automated repair, a single justification for an entailment can be of enormous benefit for ontology debugging. Indeed, the availability of a single justification can draw an ontology engineer's attention to the source of a problem with their ontology, and can be enough to allow them to understand the problem and perform a meaningful manual repair on their ontology. Being able to compute a single justification for an entailment is therefore an important task.

The Reasoner-Coupling Axis: Black-Box versus Glass-Box Algorithms As far as explanation generation is concerned, the basic distinction between glass-box and black-box algorithms was introduced by Parsia [PSK05]. The categorisation is based entirely on the part played by reasoning during the computation of justifications. In essence, justifications are computed as a *direct consequence of reasoning* in glass-box algorithms, whereas they are *not* computed as a direct consequence of reasoning in a black-box algorithm. In this sense, glass-box algorithms are tightly interwoven with reasoning algorithms, whereas black-box algorithms simply use reasoning to compute whether or not an entailment follows from a set of axioms.

Coupled together, the two axes mean that there are three important combinations of algorithms for computing *all* justifications for an entailment:

- **All-Black-Box with a Single-Black-Box Subroutine** A black-box algorithm for computing all justifications uses a black-box algorithm for computing a single justification as a sub-routine. Such algorithms [Jun04, KPS05, Sch05b, FS05, Kal06, SH07, JHQ^{+}09] are sometimes known as *pure black-box* algorithms.
- **All-Black-Box with a Single-Glass-Box Subroutine** A black-box algorithm for computing all justifications uses a glass-box algorithm for computing *single* justifications as a sub-routine. Such algorithms [Kal06, KK07, Sun09] are sometimes known as *hybrid black-box glass-box* algorithms.
- **All-Glass-Box (with a Single-Glass-Box Subroutine)** A glass-box algorithm is used for computing *all* justifications. Such algorithms [SC03, KPSH05, LMPB06, MLBP06, Lam07, LSPV08] are sometimes known as *pure glass-box* algorithms. Note that in this case, the distinction along the single-all axis is somewhat blurred.

It should be noted that, in this thesis, the term glass-box algorithm is also used to describe a glass-box based algorithm that computes *small sets of axioms*, which *may* be justifications or may be slightly larger than justifications.

An advantage of black-box algorithms is that they can be easily and robustly implemented [Kal06]. This is because a black-box implementation simply relies on the availability of a reasoner that implements a sound and complete reasoning procedure for the logic in question. Black-box algorithms work for any monotonic logic for which entailment checking is decidable. A perceived disadvantage of black-box algorithms is that they can be inefficient and impractical due to a potentially large search space [Stu08].

The perceived advantage of glass-box algorithms is that, at least for computing single justifications, justifications get computed "quickly" and "for free" as part of the reasoning process. A perceived disadvantage of glass-box algorithms is that, in contrast to black-box algorithms, implementation is tricky, and adapting an existing reasoner implementation so that it supports glass-box justification finding is highly non-trivial [Kal06].

In what follows, algorithms for computing single justifications are first discussed, followed by a presentation of algorithms for computing all justifications.

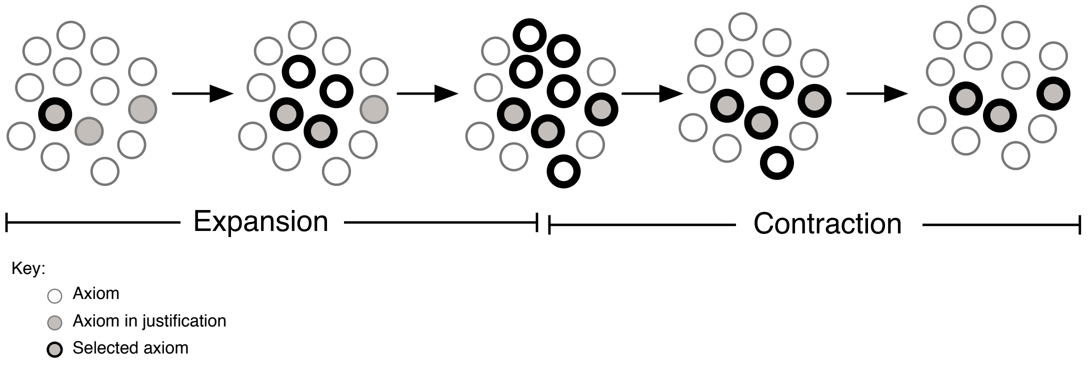


Figure 3.1: A Depiction of a Black-Box Expand-Contract Strategy

3.2 Black-Box Algorithms for Computing Single Justifications

The basic idea behind a black-box justification finding algorithm is to systematically test different subsets of an ontology in order to find one that corresponds to a justification. As depicted in Figure 3.1, subsets of an ontology are typically explored using an "expand-contract" strategy. In order to compute a justification for $\mathcal{O} \models \eta$, an initial, small, subset $\mathcal{S}$ of $\mathcal{O}$ (represented by circles with thick black borders in Figure 3.1) is selected. The axioms in $\mathcal{S}$ are typically the axioms whose signature has a non-empty intersection with the signature of η, or axioms that "define"[2] terms in the signature of η. A reasoner is then used to check if $\mathcal{S} \models \eta$, and if not, $\mathcal{S}$ is expanded by adding a few more axioms from $\mathcal{O}$. This *incremental expansion* phase continues until $\mathcal{S}$ is large enough so that it entails η. When this happens, either $\mathcal{S}$, or some subset of $\mathcal{S}$, is *guaranteed* to be a justification for η. At this point $\mathcal{S}$ is gradually *contracted* until it is a *minimal* set of axioms that entails η i.e. a justification for η in $\mathcal{O}$.

In some black-box algorithms the expand phase may be trivial, or "empty", where $\mathcal{S}$ is immediately expanded to all input axioms. In this situation it is the contraction phase which "does all the work". An example of such a strategy is presented in Algorithm 3.1. In this algorithm a set of axioms $\mathcal{S}$ is initialised (expanded) with all of the axioms in $\mathcal{O}$ so that $\mathcal{S} \models \eta$. $\mathcal{S}$ is then pruned one axiom at a time, so that for each $\alpha \in \mathcal{S}$, if $\mathcal{S} \setminus \{\alpha\} \models \eta$, then $\mathcal{S} = \mathcal{S} \setminus \{\alpha\}$. This process terminates when all axioms $\alpha \in \mathcal{S}$ have been examined, at which point

[2]For example, the axiom $A \sqsubseteq B$ defines the class name A

$\mathcal{S}$ corresponds to a justification for $\mathcal{O} \models \eta$. It is easy to see that this simple algorithm requires $n = |\mathcal{O}|$ entailment tests.

Algorithm 3.1 ComputeSingleJustification

```
ComputeSingleJustificationSimple($\mathcal{O}$, $\eta$)
1: $\mathcal{S} \leftarrow \mathcal{O}$
2: for each $\alpha \in \mathcal{S}$ do
3:     if $\mathcal{S} \setminus \{\alpha\} \models \eta$ then
4:         $\mathcal{S} = \mathcal{S} \setminus \{\alpha\}$
5:     end if
6: end for
7: return $\mathcal{S}$
```

While the above algorithm might be effective for small ontologies, it is easy to imagine that it would be impractical for moderately large ontologies that contain a few hundred axioms or more. This is because the number of entailment tests grows linearly with the number of axioms. Even for weak logics such as $\mathcal{EL}$, for which the complexity of entailment testing is polynomial in the size of the input, entailment testing can be prohibitively time consuming. Indeed, in [Sun09], Suntisrivaraporn showed that naive algorithms such as Algorithm 3.1 are impractical for large ontologies such as SNOMED. For this reason, there are various optimisations that aim to prune the search space. These optimisations essentially help to minimise the number of entailment tests required in both the expansion and contraction phases, and aim to minimise the difference in size between the expanded set of axioms and the final justification size.

3.2.1 Expansion Phase Optimisations

The main idea behind expansion phase optimisations is to quickly find a set of entailing axioms, which is hopefully not much larger than the final justification. In other words, the ultimate aim is to find a set of axioms that is a justification, or a small superset of a justification, with a small number of entailment tests. There are two main techniques for doing this: (1) Expansion by Selection Function, and (2) Expansion by Modularisation.

Incremental Expansion By Selection Function

Given an ontology $\mathcal{O} \models \eta$, and a, possibly empty, set of axioms $\mathcal{S} \subseteq \mathcal{O}$, a selection function, σ can be defined as follows (where $\mathcal{P}(\mathcal{O})$ is the Powerset of $\mathcal{O}$):

$$\sigma_{\mathcal{O},\eta} \colon \mathcal{P}(\mathcal{O}) \rightarrow \mathcal{P}(\mathcal{O})$$

Essentially, given some subset $\mathcal{S} \subseteq \mathcal{O}$, a selection function σ chooses another set of axioms $\mathcal{S}' \subseteq \mathcal{O}$. It is usually the case that selection functions are monotonic, that is, if $\mathcal{S} \subsetneq \mathcal{O}$ then $\mathcal{S} \subsetneq \mathcal{S}'$.

The above notion of a selection function is similar to the notion of a *relevance based selection function* introduced by Ji et al. [JQH09]. The difference, is that their selection function is parameterised by $\mathbb{N}$, elements of which are used to denote a "step" and ultimately a "degree of relevance" in the expansion phase. In the selection function here, the steps are not explicit, but are simply a consequence of the definition of the function.

As can be imagined, the design of a selection function can vary from one implementation to another, and more importantly can be based on the logic in question. The most basic selection functions simply select axiom sets based on the *signature* of the axioms and entailment in the input [KPHS07, SQJH08, JQH09]. For example, given $\mathcal{O} \models \eta$ and $\mathcal{S} \subseteq \mathcal{O}$, such a selection function $\sigma(\mathcal{S}, \mathcal{O}, \eta)$ outputs

$$\{\alpha \mid \mathsf{signature}(\alpha) \cap (\mathsf{signature}(\eta) \cup \mathsf{signature}(S)) \neq \emptyset\}$$

For example, consider

$$\mathcal{O} = \{A \sqsubseteq B,\ B \sqsubseteq C,\ B \sqsubseteq D,\ A \sqsubseteq \neg D,\ A \sqsubseteq E\} \models A \sqsubseteq \bot\ .$$

In the first step, given the entailment of interest $A \sqsubseteq \bot$, a selection function would choose every axiom with A in its signature and add this to $\mathcal{S}$ so that

$$\mathcal{S} = \{A \sqsubseteq B,\ A \sqsubseteq \neg D,\ A \sqsubseteq E\}\ .$$

Since $\mathcal{S}$ does not entail $A \sqsubseteq \bot$ the selection function would then choose further axioms from $\mathcal{O}$ based on $\mathcal{S}$ to give

$$\mathcal{S}' = \{A \sqsubseteq B,\ B \sqsubseteq C,\ B \sqsubseteq D,\ A \sqsubseteq \neg D,\ A \sqsubseteq E\}\ .$$

At this point $\mathcal{S} \models \eta$ and the algorithm would enter the contraction phase.

The main perceived problem with such simple selection functions is that they quickly "blow up". That is, for real ontologies, it only takes a few steps before the whole ontology is selected. To combat this problem, selection functions that exploit particular logic fragments and the syntax of axioms have been designed. For example, in [Sch05b, SHCvH07], a selection function for *unfoldable* $\mathcal{ALC}$ is defined which directly mimics the lazy unfolding procedure [Hor97] that used in reasoning algorithms for unfoldable knowledge bases [Neb90, Baa91]. In [Hv06] the notion of *concept relevance* is introduced, which takes into consideration the left and right hand sides of axioms of the form $C \sqsubseteq D$, $C \equiv D$ and $\mathsf{DisjointWith}(C, D)$. A small amount of empirical work detailed in [Hv06] seems to indicate that the concept relevance selection function is beneficial over the naive selection function.

In essence, incremental expansion aims to find the entailment of interest in a subset of an ontology that is hopefully much smaller than the ontology itself. The benefit of doing this is that (1) Entailment tests are typically faster for smaller sets of axioms; and (2) A smaller input to the contraction stage can result in fewer entailment tests in that phase.

Expansion by Modularisation

As discussed in the preliminaries chapter, there are now various well defined notions of entailment preserving modules for expressive Description Logics such as $\mathcal{SHOIQ}$. In terms of computing justifications there have been numerous publications which show that modularisation can significantly boost the performance of justification finding algorithms [Sun08, BPS07, BS08, SQJH08, Nor01].

Computing a justification, or all justifications, with respect to a module $\mathcal{M}$ of $\mathcal{O}$ rather than $\mathcal{O}$ can be a helpful optimisation because, in practice, semantic based modularisation algorithms tend to compute modules for real ontologies that are much smaller than the ontology itself. For example, on average, the size of a *strong subsumption module* [BS08] in the medical ontology SNOMED is around 53 axioms in comparison to the size of the ontology, which is over 380,000 axioms [BS08]. The benefits of computing justifications in a module $\mathcal{M}$ of an ontology $\mathcal{O}$ (when $\mathcal{M}$ is much smaller than $\mathcal{O}$) are therefore:

1. Entailment checking with respect to $\mathcal{M}$ can be much faster than entailment checking with respect to the whole ontology. This is simply due to the

reduction in size from the ontology to the module—the time required for a reasoner to load and preprocess $\mathcal{M}$ is less than that for $\mathcal{O}$. This can be beneficial in both the expansion and contraction phases of a justification finding algorithm.

2. The contraction phase has less work to do when pruning axioms from $\mathcal{M}$ compared to $\mathcal{O}$. This means less entailment tests are required to prune $\mathcal{M}$ compared to $\mathcal{O}$ which boosts overall performance.

3. The module provides a "hard boundary" for sloppy or promiscuous selection functions. As explained above, selection functions hopefully choose small entailing subsets of $\mathcal{O}$, however if a naive selection function is used, it could result in a blow up of the axioms after just a few selection steps. Modules can essentially contain this blow up and enable the contraction phase to operate on a smaller subset of $\mathcal{O}$ than would otherwise be the case.

In summary, the ability to compute small, conservative extension like, modules for entailments in ontologies, is attractive from the point of view of optimising justification finding. The primary reason for this is due to the fact that for real world large ontologies: (1) Modules can be computed quickly and efficiently. Computing modules does not have a negative impact on the time take to compute justifications; (2) Modules are generally much smaller than their corresponding ontologies. This means that entailment checking performance can be significantly boosted.

3.2.2 Contraction Phase Optimisations

Given a set of axioms $\mathcal{S}$, where $\mathcal{S} \models \eta$ there are two important optimisations that can help overcome the performance problems of the naive contraction strategy presented in Algorithm 3.1.

Sliding Window

In [Kal06, KPHS07], Kalyanpur (et al.) use a "sliding window" technique. This operates in a similar fashion to the simple strategy described in Algorithm 3.1, except that instead of choosing one axiom α for removal in each step, a set of axioms $\mathcal{S}'' \subset \mathcal{S}$ is removed from $\mathcal{S}$. The advantage here is that if $\mathcal{S} \setminus \mathcal{S}'' \models \alpha$ then $n = |\mathcal{S}''|$ axioms can be discarded, saving $n - 1$ entailment tests over the simple contraction strategy in Algorithm 3.1. The sliding window technique is also used by Moodley [Moo10] for computing *root justifications*[3]. The advantage of the sliding window technique is that it is extremely easy to implement. However, although some empirical evidence suggests a window size of $n = 10$ provides good performance [KPHS07], it is not clear if this is the best window size for achieving optimal performance, or if the best window size varies depending on ontology structure.

Divide and Conquer

An improvement over the sliding window technique described above, is provided by a classic "divide and conquer" based strategy. Given and ontology $\mathcal{O}$, where $\mathcal{O} \models \eta$, $\mathcal{O}$ is split into two halves, $\mathcal{O}_1$ and $\mathcal{O}_2$. If $\mathcal{O}_1$ is sufficient to entail η then $\mathcal{O}_2$ is discarded and vice-versa. If neither $\mathcal{O}_1$ or $\mathcal{O}_2$ separately entail η then $\mathcal{O}_1$ is split into two halves and one of these halves is combined with $\mathcal{O}_2$ to see if η is entailed by this half and $\mathcal{O}_2$. This process of halving and combining halves is continues until a justification for η is obtained.

The divide and conquer strategy is a classic technique in computer science and is widely used as the basis for searching, sorting and merging algorithms. In the context of black-box justification finding, it was originally proposed by Junker [Jun01, Jun04] as part of his QUICKXPLAIN algorithm. Junker's work did not feature any empirical evaluation of his algorithms. However, at least two empirical evaluations of the divide and conquer algorithm in the context of justification finding were later carried out. First, Friedrich and Shchekotykhin [FS05] investigated the benefits of this algorithm on a small collection of 8 ontologies, and then Suntisrivaraporn carried out an empirical investigation on SNOMED [Sun09]. Finally, Shchekotykhin et al [SFJ08] carried out a comparison between the sliding window strategy used by Kalyanpur et al. [KPHS07] and the dividing and conquer strategy, and found that the divide and conquer strategy performed noticeably better.

[3]Given an ontology $\mathcal{O}$, a set of axioms $\mathcal{S} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ such that $\mathcal{O} \models \mathcal{S}$ and a set $\mathfrak{J} = \{\mathcal{J} \mid \mathcal{J}$ is a justification for $\mathcal{O} \models \alpha_i$ where $\alpha_i \in \mathcal{S}\}$ then $\mathcal{J} \in \mathfrak{J}$ is a root justification for $\mathcal{S}$ with respect to $\mathcal{O}$ if there is no $\mathcal{J}' \in \mathfrak{J}$ such that $\mathcal{J}' \subsetneq \mathcal{J}$. If $\mathcal{J}$ is not a root justification then it is a *derived justification* [MMV10].

3.3 Computing All Justifications Using Model Based Diagnosis Techniques

The above techniques deal with black-box computation of single justifications for entailments in ontologies. When formulating a repair plan for an entailment, or attempting to understand an entailment, it is usually necessary to compute *all* justifications for that entailment. This can be achieved using black-box techniques for finding single justifications in combination with techniques that are borrowed from the field of *model based diagnosis* [DH88, HCK92].

Model Based Diagnosis is the overarching name for the process of computing *diagnoses* for a *faulty system.* Briefly, a *diagnosis* is a subset minimal set of *components* from a faulty system, that if replaced would *repair* the system so that it functions as intended. In this context the term diagnosis is taken from the medical terminology, wherein it refers to both the process and the outcome of identifying a disease or other medical problem from its symptoms. The term *model* refers to the fact that a model is used to obtain the *predicted* behaviour of of a system, which is then compared the *observed* behaviour of the system. Any discrepancy between the predicted and observed behaviours indicates a fault in the system. Model based diagnosis techniques have a wide variety of applications, from diagnosing sets of faulty gates and connections in circuits [Gen84] through to debugging spreadsheets [JE10].

The basic idea is that the manifestation of faults in a system is put down to the bad interaction between different sets of components in that system. A faulty set of components from a system is known as a *conflict set* if that set of components exhibits a fault that is exhibited in the overall system. A conflict set is a *minimal conflict set* if no proper subset of it is a conflict set. Given a set of conflict sets for a system, a *hitting set* for these conflict sets contains *at least one* component from each conflict set. A hitting set is a *minimal hitting set* if no proper subset of it is a hitting set. In model based diagnosis, a minimal hitting set is known as a *diagnosis*, and replacing the set of components which corresponds to a diagnosis eradicates the fault from the system.

3.3.1 The Relationship Between Model Based Diagnosis and Justification Based Explanation

Given the above notions from the field of Model Based Diagnosis, it should be clear that there is a direct parallel with the notions from justification based explanation. Indeed, an ontology corresponds to a system, an entailment in the ontology corresponds to a fault in the system, a justification corresponds to a minimal conflict set, and a diagnosis corresponds to a minimal repair of the ontology.

Since the field of Model Based Diagnosis concerns the study of algorithms for computing diagnoses, and there is a direct relationship between diagnoses and repairs for ontologies, the algorithms used in model based diagnosis may be borrowed for the purposes of computing repairs for entailments in ontologies, and in doing this, may be used for the purposes of computing *all justifications* for an entailment in an ontology.

A particularly pertinent algorithm from the field of Model Based Diagnosis is "Reiter's Hitting Set Tree (HST) Algorithm" [Rei87, GSW89]. Given a faulty system, Reiter's algorithm computes all minimal diagnoses for the system. This algorithm essentially constructs a finite tree whose nodes are labelled with minimal conflict sets (justifications), and whose edges are labelled with components (axioms) from a system (ontology). In doing this the algorithm finds *all minimal hitting sets*, which represent diagnoses for the conflict sets in the system. Given the correspondence between conflict sets and justification, diagnoses and repairs, it is possible to use Reiter's algorithm to find all repairs, and in doing this find all justifications for an entailment in an ontology (proof in [KPHS07]).

3.3.2 A Hitting Set Tree Approach to Computing All Justifications for an Entailment

The following example provides a high level illustration of how a Hitting Set Tree can be used to compute all justifications for an entailment in some ontology. A more formal description of the procedure will be given later in Algorithm 4.1 on Page 81.

Before continuing, it is necessary to make the notion of a hitting set tree more concrete. Given $\mathcal{O} \models \eta$, a hitting set tree for η in $\mathcal{O}$ is a finite tree, an example of which is shown in Figure 3.2, which consists of nodes labeled with justifications for $\mathcal{O} \models \eta$ and edges labelled with axioms contained in $\mathcal{O}$. Each non-leaf node v is connected to a successor node v' via an edge labelled with an axiom α such that α is in the label of v but *not* in the label of v'. The label of v' may be the empty set, in which case v' must be a leaf node. Moreover, for any node v'', the set of axioms that label the path from v'' to the root of the tree does *not* intersect with the justification that labels v''.

The construction of a hitting set tree may take place in a breadth first or depth first manner. Which ever is used, the principles and rules governing the generation and labelling of edges and nodes in the tree are the same: At any point, when extending the tree from a node v, to a new node v' the basic procedure is:

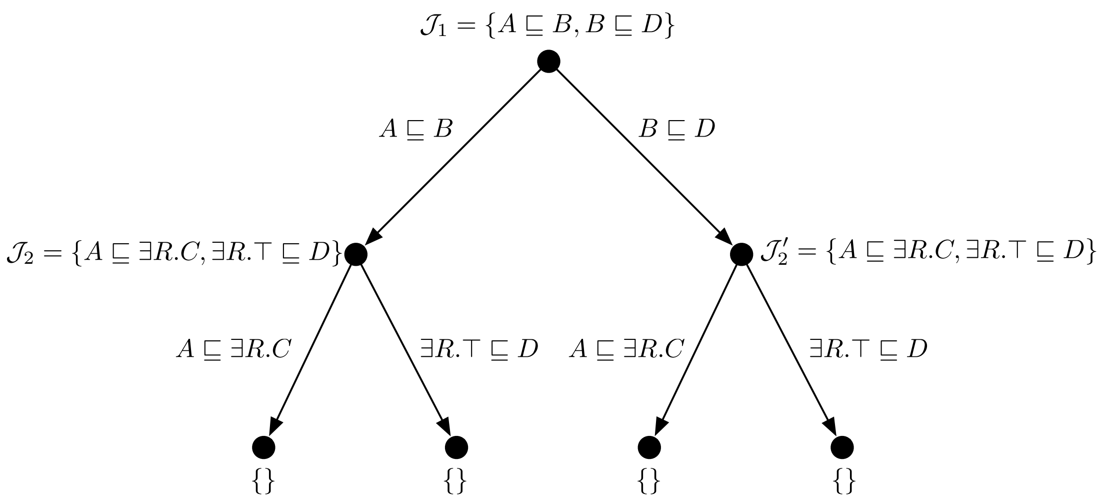
1. Choose an axiom α that is in the label of v but does not label an edge that connects v to an existing successor node.
2. Set $\mathcal{S}$ to be the union of $\{\alpha\}$ and the set of axioms that label the edges that form the path from v to the root of the tree. Remove $\mathcal{S}$ from $\mathcal{O}$ to give $\mathcal{O}'$.
3. If $\mathcal{O}' \models \eta$ then compute a justification $\mathcal{J}$ for η with respect to $\mathcal{O}'$. If $\mathcal{O}' \not\models \eta$ then set $\mathcal{J} = \emptyset$.
4. Create a fresh node v' and set the label of v' to be $\mathcal{J}$.
5. Add the edge $e = \langle v, v' \rangle$ to the tree and label e with $\mathcal{J}$.
6. Add the axioms in $\mathcal{S}$ back into $\mathcal{O}$.

For a more concrete example consider the following ontology

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B\\ &B \sqsubseteq D\\ &A \sqsubseteq \exists R.C\\ &\exists R.\top \sqsubseteq D\} \models A \sqsubseteq D\end{aligned}$$

which entails $\eta = A \sqsubseteq D$. A hitting set tree for $\mathcal{O} \models A \sqsubseteq D$ is shown in Figure 3.2. Taking the root node, which is labelled with J_1, the hitting set tree was extended to the left hand side by removing $A \sqsubseteq B$ from $\mathcal{O}$ and computing justification for $\mathcal{O} \setminus \{A \sqsubseteq B\} \models A \sqsubseteq D$. In this case, the justification $\mathcal{J}_2$ was found. The left hand side successor node of the root node was therefore labelled with $\mathcal{J}_2$ and its connecting edge labelled with $A \sqsubseteq B$. Similarly, the

Figure 3.2: An Example of a Hitting Set Tree


$\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq D\}$
$A \sqsubseteq B$
$B \sqsubseteq D$
$\mathcal{J}_2 = \{A \sqsubseteq \exists R.C, \exists R.\top \sqsubseteq D\}$
$\mathcal{J}_2' = \{A \sqsubseteq \exists R.C, \exists R.\top \sqsubseteq D\}$
$A \sqsubseteq \exists R.C$
$\exists R.\top \sqsubseteq D$
$A \sqsubseteq \exists R.C$
$\exists R.\top \sqsubseteq D$
$\{\}$
$\{\}$
$\{\}$
$\{\}$

bottom right hand successor to the node labelled with $\mathcal{J}_2'$ and whose successor edge is labelled with $\exists R.\top \sqsubseteq D$ was generated by considering $\mathcal{O} \setminus \mathcal{S}$ where $\mathcal{S} = \{B \sqsubseteq D,\ \exists R.\top \sqsubseteq D\}$ ($\exists R.\top$ plus the label of the path to the root) and noting that $\mathcal{O} \setminus \mathcal{S}$ does not contain a justification for $A \sqsubseteq D$. A fresh successor node was therefore generated and labelled with the empty set, with the successor edge label being set as $\exists R.\top \sqsubseteq D$.

When no more successor nodes can be generated the hitting set tree is complete. At this point, *all* justifications for $\mathcal{O} \models \eta$ occur as labels of nodes in the tree. Additionally, all minimal repairs (diagnoses) for $\mathcal{O} \models \eta$ are contained as leaf-root paths in the tree.

3.3.3 Model Based Diagnosis Optimisations

The above description of a hitting set tree and illustrative example do not take into consideration any optimisations. In order to achieve acceptable performance it is necessary to consider two important optimisations: (1) *Early Path Termination*, and (2) *Justification Reuse*, which are detailed below:

Early path termination

In the unoptimised version of the algorithm a node can be extended with successor edges provided there is an axiom in its label which does not already label one of the edges of its successors. Nodes that can be extended are referred to as *open* nodes. Nodes which cannot be extended are referred to as *closed* nodes. Obviously, leaf nodes cannot be extended so they are closed nodes. When optimisations come into play, it can be the case that non-leaf nodes which would otherwise be labelled with a non-empty set are marked as closed. In this case, the path from the closed node to the root node is said to have been subject to *early termination*, and its path to the root has been *early terminated*. Detecting conditions for and enacting early termination is a key optimisation for the hitting set tree algorithm and works as follows: If a hitting set tree T contains an open node v_1, which has a path P_1 to the root of the tree, and there is some other open node v_2 which has a path P_2 to the root of the tree, then if the label set of P_1 is equal to the label set of P_2 then v_2 may be marked as closed and need not be explored further. The reason for this is that the descendent nodes of v_2 would be labelled with exactly the same justifications as the descendent nodes of v_1. This is because the labels (justifications) of v_2 descendants are calculated with respect to some subset of $\mathcal{O}\setminus\mathcal{S}_2$ and the labels of v_1 descendants are calculated with respect to $\mathcal{O}\setminus\mathcal{S}_1$, where $\mathcal{S}_2$ and $\mathcal{S}_1$ are the label sets of the paths to the root of v_2 and v_1 respectively (which are the same). Hence, only one of P_1 or P_2 needs to be explored further to find the remaining justifications.

Justification reuse

The second most important optimisation is justification reuse. In the unoptimised hitting set tree construction algorithm, a justification is *computed* (with a call to a black-box or glass-box subroutine) for each node v that is added to the tree. The justification, which is used to label v, is computed with respect to $\mathcal{O}\setminus\mathcal{S}$ where $\mathcal{S}$ is the set of labels on the path from v to the root. If the tree contains some other node v' which is labelled with a justification $\mathcal{J}$, and $\mathcal{S}$ does *not* intersect with $\mathcal{J}$ then $\mathcal{J}$ can be used to label the node v. The reason for this is that it must be the case that $\mathcal{J} \subseteq \mathcal{O} \setminus \mathcal{S}$ (since $\mathcal{J} \subseteq \mathcal{O}$ and $\mathcal{S} \cap \mathcal{J} = \emptyset$), hence $\mathcal{J}$ could be computed as a justification that could label v. Justification reuse can save a lot of unnecessary calls to a subroutine for computing single justifications.

3.3.4 Complexity Issues

One of the issues with computing all justifications for an entailment is that the size of the hitting set tree can, in the worst case, be exponential in the size of the set of justifications. This worst case scenario occurs when there are non-overlapping justifications for an entailment. For example, consider the set of justifications $\big\{\{\alpha'_1, \alpha''_1\}, \ldots, \{\alpha'_n, \alpha''_n\}\big\}$. The complete hitting set tree contains $2^{(n+1)} - 1$ nodes, with all paths from leaf to root being of length n. The basic intuition for this, is that there is an exponential number of repairs for the ontology which contains the justifications—adding a new justification doubles the number of repair options. Since all possible repairs need to be taken into consideration when finding all justifications, and all possible repairs are reflected in the structure of the hitting set tree, the size of the tree is exponential in the number of justifications.

Despite this high worst case complexity, as will be seen later, it is possible to compute large numbers of justifications for entailments in real ontologies. For example, in one ontology, the maximum number of justifications tops 800 (see Experiment 1 beginning on Page 98). These numbers are clearly much larger than the number of justifications that would be possible to compute in the worst case scenario, where the hitting set tree contains in the order of 2^n nodes for n justifications. The reason for this is that the complexity of the algorithm depends upon the degree to which justifications *overlap*. The higher the overlap, the more effective early path termination is. In some cases, for a given number of justifications, overlap can lead to an exponential decrease in the number of nodes in the hitting set tree. Fortunately, as will be seen later, it is typically the case that, in real world ontologies either the number of justifications for an entailment is small enough to allow all of them to be computed, or the number of justifications is large, but they overlap enough such that all of them can be computed.

3.4 Computing Justifications Using Glass-Box Techniques

As mentioned previously, glass-box justification finding algorithms are characterised by the fact that they are tightly interwoven with the actual reasoning algorithms. They compute some, or all justifications, as part of reasoning. There have been numerous glass-box algorithms that have been developed for different DLs. For example, Schlobach and Cornet's work [SC03, SHCvH07] was concerned with Unfoldable-$\mathcal{ALC}$. Meyer et al. and then Lam et al. extended this to $\mathcal{ALC}$ with general concept inclusions [LMPB06, MLBP06, LSPV08], Kalyanpur et. al provided an algorithm for the highly expressive DL $\mathcal{SHOIN}$ [Kal06], and Suntisrivaraporn and Baader described a glass-box approach for $\mathcal{EL}$ and $\mathcal{EL}++$ [BPS07, BS08]. All of these approaches are essentially based on what is known as *tracing*. This is a process which involves tagging reasoning structures, such as assertions in a tableau completion-ABox, with labels that ultimately identify the original axioms in an input ontology that caused the production and expansion of these structures.

In the context of the current breed of glass-box justification finding implementations, most them can be related back to some seminal work by Baader and Hollunder [BH95]. In this work Baader and Hollunder were primarily concerned with embedding Reiter's default logic [Rei80] into $\mathcal{ALCF}$ knowledge bases, which required the ability to compute minimally inconsistent, and maximally consistent *ABoxes*. To do this, Baader and Hollunder extended the tableau based consistency algorithm for $\mathcal{ALC}$ ABoxes, augmenting it with a "tracing" mechanism. In essence, tracing adds monotone boolean formulae as labels to assertions in completion ABoxes, which "pinpoint" the input axioms which caused their generation. For a given entailment the output is a pinpointing formula that can be used to identify all justifications for that entailment. Before the description of tracing is presented, it is worth noting that, in recent work [nal09, BP10], Peñaloza and Baader have taken a more general approach to tracing, where they provide a general specification of tableau reasoning algorithms, and then present a description and analysis of tracing at this more general level rather than at the level of a specific DL.

3.4.1 Tracing

Given an $\mathcal{ALC}$ ABox $\mathcal{A}$, the $\mathcal{ALC}$ tableau algorithm for ABox consistency generates a set of completion $\mathcal{ALC}$ ABoxes $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$. Each of these completion ABoxes contains assertions that are derived by the recursive application of the $\mathcal{ALC}$ tableau rules for ABox consistency [SSS91], initially on $\mathcal{A}$ and then on each generated ABox $\mathcal{A}_i$ (See Section 2.2 on Page 40 for more details on Tableau Algorithms). The tracing aspect of the work begins by introducing a propositional variable for each ABox assertion in the input ABox $\mathcal{A}$. Each assertion in the set of completion ABoxes, which is derived by the tableau algorithm, is then labelled with a monotone boolean formula, which represents how that assertion was derived. For example, given the ABox $\mathcal{A} = \{(C \sqcap D)(a)^p\}$, where $(C \sqcap D)(a)$ is labelled with the propositional variable p, the ABox $\mathcal{A}_1 = \{(C \sqcap D)(a)^p, C(a)^p, D(a)^p\}$ is derived by application of the $\sqcap$-rule. Notice how each of the derived assertions $C(a)^p$ and $D(a)^p$ is labelled p. In the case where multiple assertions are used to derive an ABox assertion, a conjunction of proposition variables is used to label the derived assertion. For example, given $\mathcal{A} = \{(\exists R.C)(a)^p, (\forall R.D)(a)^q\}$, the ABox $\mathcal{A}_1 = \{(\exists R.C)(a)^p$, $(\forall R.D)(a)^q$, $R(a,b)^p, D(b)^{p\wedge q}, \ldots\}$ is derived, where b is a fresh individual name. Here, both $R(a,b)^p$ and $(\forall R.C)^q$ are used to derive the assertion $D(b)^{p\wedge q}$. Conversely, since there may be more than one way to derive an assertion, disjunctions can be present in the labelling formulae. For example, if $\mathcal{A} = \{(C \sqcap D)(a)^p, (C \sqcap E)(a)^q\}$ then the assertion $C(a)^{p\vee q}$ can be derived from the assertions labelled with p or q, and so it gets the label $p \vee q$.

When the tableau algorithm terminates, the labels on assertions in the set of ABoxes $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ can be used to determine which assertions in original ABox $\mathcal{A}$ are responsible for the inconsistency of $\mathcal{A}$. For a given ABox $\mathcal{A}_i$, the labels result in what Baader and Hollunder call a *clash formula*—also known as a *pinpointing formula*. This is obtained by taking each clash $\{A(a)^p, \neg A(a)^q\} \subseteq \mathcal{A}_i$ and conjoining the formulae of the positive and negative assertion to give $p \wedge q$. For each completion ABox $\mathcal{A}_i$, every such conjunction is combined into a disjunction, which represent all of the clashes that completion ABox. Finally, each of these $\mathcal{A}_i$ disjunctions is combined into one large conjunction that represents the final clash formula for input ABox $\mathcal{A}$.

Given the set of propositional variables Q that correspond to the assertions in $\mathcal{A}' \subseteq \mathcal{A}$, it holds that there is an assignment v on these variables, where $v \colon Q \mapsto \{\mathsf{true}, \mathsf{false}\}$, that makes the clash formula for $\mathcal{A}$ true if and only if $\mathcal{A}'$ is inconsistent. Each minimal satisfying assignment v of Q for the corresponding $\mathcal{A}'$, where minimal means subset minimal with respect to the variables in Q that are assigned the value true, identifies $\mathcal{A}'$ as a subset minimal set of assertions that is responsible for $\mathcal{A}$ being inconsistent [BH95]. In essence, the problem of finding minimal sets of assertions that make $\mathcal{A}$ inconsistent is reduced to finding minimal assignments that make the clash formulae for $\mathcal{A}$ true.

Finally, Baader and Hollunder point out that the problem of finding minimal valuations is an NP-Complete problem, and that if the clash formulae are in conjunctive normal form (CNF) then this corresponds to the *hitting set* problem [GJ79]. If the clash formulae are in disjunctive normal form (DNF) then the problem is in PTime, but transforming a monotone boolean formula into DNF can result in an exponential blow in the size of the original formula.

3.4.2 Extensions of Tracing to More Expressive Logics

Although the work by Baader and Hollunder was not driven by the end need for computing explanations per se, as can be seen, their so-called *tracing technique*, does indeed compute justifications for entailments. In some way, all subsequent work on glass-box justification finding, including work that does not directly compute a clash formula, can be related back to their tracing technique. Indeed, the basic principles of Baader and Hollunder's tracing technique, can be used to find just *one* justification, *some* justifications, or *all* justifications for an entailment depending on whether or not the tableau algorithm is run until every ABox $\mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is *complete*.

As mentioned previously, Schlobach and Cornet were the first to apply a glass-box tracing technique to computing justifications. Rather than working with ABoxes, their technique [SC03] was used with Unfoldable-$\mathcal{ALC}$ TBoxes, which they chose over $\mathcal{ALC}$ with general concept inclusions in order to simplify the presentation and computational properties of their algorithms. The next steps came when the technique was expanded by Meyer et al. [MLBP06] to deal with acyclic $\mathcal{ALC}$ with general concept inclusion axioms. This was later extended by Lam et al. [Lam07, LPSV06] to work in the context of fine-grained justifications.

In all of the cases above, glass-box tracing is used to compute *all* justifications for an entailment. However, in other work, notably work on more expressive DLs, tracing has only been used for computing *single* justifications. To understand why this is the case it is necessary to consider how tableau algorithms are implemented in optimised reasoners. Recall that an ABox is *complete* if no more tableau completion rules are applicable to it, and that this is different to the case where no more rules are applied to an ABox because is *closed* i.e. contains at least one clash. In the case where the tableau algorithm is run until every $\mathcal{A}_i$ is *complete*, and every $\mathcal{A}_i$ contains *all clashes*, then tracing can be used to find *all* justifications for an entailment as described above. However, if the tableau algorithm is only run until the *first clash* is found in each $\mathcal{A}_i$ (as is the case with all practical and optimised reasoners), then the tracing technique will find *some* justifications i.e. at least *one*, but it is not guaranteed to find *all* justifications for the entailment in question. This is the case in [KPSH05, Kal06], where Kalyanpur describes a glass-box tracing algorithm for the DL $\mathcal{SHOIN}$, which can be used to find a single justification. The tracing algorithm described in Kalyanpur's work also differs from Baader and Hollunder's tracing algorithm in another way: it labels assertions with sets of pointers to axioms rather than labelling them with a monotone boolean formulae. Given a clash in each completion ABox the algorithm unions the sets of pointers and then computes a justification from these pointers. In work by Suntisrivaraporn [Sun08], Baader and Hollunder style tracing is described for the lightweight Description Logic $\mathcal{EL}+$. However, the actual implementation that is described only uses glass-box techniques to compute a single justification. The reason for this is that with the $\mathcal{EL}$ family of DLs the primary concern is polynomial time reasoning, and the glass-box tracing algorithm requires the entailment checking of monotone boolean formulae which is an NP-Complete problem.

When glass-box tracing is only used for computing single justifications it can be combined with model-based diagnosis techniques for computing all justifications. Kalyanpur et al. [Kal06], Kremen and Kouba [KK07], Suntisrivaraporn [Sun09] all use this approach.

3.4.3 Inexact Glass-Box Tracing

In the glass-box tracing algorithm for computing a single justification, which is described by Kalyanpur [Kal06], and the glass-box tracing described by Suntisrivaraporn [Sun09], the set of axioms returned may be a superset of a justification. In the case of the $\mathcal{EL}+$ algorithm this is caused by the normalisation procedure, which may normalise many axioms into a single axiom. In the case of the $\mathcal{SHOIN}$ tableau algorithm the reason is slightly less straight forward. In [Kal06] Kalyanpur attributes the problem of non-minimal returns by the tracing algorithm to max-cardinality restrictions and the non-determinism of the $\leq$-rule. He presents the following example to describe the problem: Consider the ontology $\mathcal{O}$ which entails $A \sqsubseteq \bot$, which has as a justification the first three axioms and the last axiom in $\mathcal{O}$

$$\begin{aligned}\mathcal{O} = \{A &\sqsubseteq \exists R.B \\ A &\sqsubseteq \exists R.(C \sqcap \neg B) \\ A &\sqsubseteq \exists R.(\neg C \sqcap \neg B) \\ A &\sqsubseteq \exists R.C \\ A &\sqsubseteq\ \leq 2R\} \models A \sqsubseteq \bot\end{aligned}$$

Now consider a completion ABox $\mathcal{A}$ which is the result of processing the first four axioms. There are four R successors of some instance of A. When the tableau algorithm comes to apply the $\leq$-rule it must merge some of these successors, since any instance of A can have at most two successors. In this example no matter which merge choice is made, all of the ABoxes that get generated from $\mathcal{A}$ contain clashes (this is ensured by the choice of fillers). In $\mathcal{SHOIN}$ tracing, when two individuals are merged, the labels also get merged. In this case, the result is a superset of a justification for $A \sqsubseteq \bot$.

Although Kalyanpur claims the problem of non-minimal returns is due to max cardinality restrictions, the real source of the problem simply lies in the ordering of the application of tableau rules in a given context. Had the tableau algorithm generated the first three successors of the instance of A due to the first three axioms in $\mathcal{O}$ and then applied the $\leq$-rule, the problem of a non-minimal return would not arise here. In essence the problem depends upon the way the tableau algorithm "unpacks" subconcepts in axioms. The following example provides an alternative, less subtle, demonstration of how non-minimal returns can arise. Consider the ontology $\mathcal{O}'$ which entails $A \sqsubseteq \bot$. The axioms have been labeled with the names p and q.

$$\begin{aligned}\mathcal{O}' = \{A &\sqsubseteq B \sqcap \exists R.(C \sqcap \neg C)^p \\ B &\sqsubseteq \bot^q\} \models A \sqsubseteq \bot\end{aligned}$$

The tableau algorithm begins by creating a fresh completion ABox, $\mathcal{A}$ that contains the assertion $A(x)$, where x is a fresh individual. Next, it unfolds A and adds the assertion $(B \sqcap \exists R.(C \sqcap \neg C))(x)^p$ to $\mathcal{A}$. The algorithm then applies the $\sqcap$-rule and adds to two further assertions $B(x)^p$ and $(\exists R.(C \sqcap \neg C))(x)^p$. Finally, the algorithm unfolds B and adds the assertion $\bot(x)^{q,p}$ to $\mathcal{A}$. Notice that $\bot(x)^{q,p}$

is labelled with both q *and* p since p caused B to be added which caused the unfolding. The glass-box algorithm then returns

$$\mathcal{S} = \{A \sqsubseteq B \sqcap \exists R.(C \sqcap \neg C),\ B \sqsubseteq \bot\}$$

which is a *superset* of a justification. In essence, the basic tableau tracing algorithm operates on concepts in assertions at the top-level and does not "peer" into subconcept in order to decide how to order rule application to ensure a minimal solution.

Rather than modifying the tableau algorithm, the simplest way to deal with this problem is to add a contraction stage after glass-box tracing which ensures return are minimal and are indeed justifications. In essence, glass-box tracing can be used as a highly optimised expansion stage that only requires one entailment check.

3.5 The Impact of Modularisation

In this thesis, no matter whether black-box or glass-box algorithms are being considered, or finding single justifications versus finding all justifications for an entailment, modularity is *always* used as an optimisation when dealing with consistent ontologies. As discussed in Section 2.4, for any input, $\mathcal{O} \models \eta$ a syntactic locality based module $\mathcal{M}$ of $\mathcal{O}$ for the signature of η contains all justifications for η with respect to $\mathcal{O}$. Computing a module is therefore used as a pre-processing step before computing *any* justifications for an entailment in the algorithms that are presented in this thesis. The actual effects of modularity based optimisation are not explicitly investigated in this thesis as there is already solid and convincing evidence in [SQJH08] (for $\mathcal{SHOIN}$ ontologies) and in [Sun09] (for $\mathcal{EL}+$ ontologies) that this kind of optimisation has a large positive impact on improving the performance of justification finding algorithms by an order of magnitude or more. It is however worth noting that, the main reason for such dramatic improvements, particularly in large ontologies of thousands of axioms or more, is that a module $\mathcal{M}$ can be significantly smaller than $\mathcal{O}$ (for example hundreds of axioms versus tens of thousands of axioms for the $\mathcal{EL}+$ version of the NCI ontology [SQJH08]). The benefits, as far as justification finding is concerned, are that the search space for finding single justifications is dramatically reduced, which means that black-box algorithms perform much better, and entailment checking performance is significantly boosted, which helps whether finding single justifications or all justifications.

3.6 Existing Empirical Evaluations

Many of the previously discussed justification finding techniques have been published along with evaluations in the form of empirical investigations. The experiments which are undertaken as part of these empirical investigations can be characterised by the ontologies used in them and their prototypical style of implementation:

Experimental Corpora Typically, the collections of ontologies used in experiments are of various styles, expressivities and sizes. The following categories summarise the various styles of ontologies used in empirical work from the literature:

- **Large well known medical ontologies**—In particular $\mathcal{EL}$ based ontologies such as SNOMED, GALEN, DICE, the Gene Ontology or early ($\mathcal{EL}$) versions of the NCI ontology. These well known "prize ontologies" are used because they fall into a size and expressivity fragment that is of interest to particular groups and communities. Used in [SQJH08, Sun09].
- **Small well known tutorial ontologies**—for example Koala, Bike, Wine, People+Pets, Chemical, Transport, Camera, which are typically used by prototypical implementations for demonstrating proof of concept. Used in [Kal06, MLBP06, KPHS07, SHCvH07, SFJ08].
- **Auto-generated ontologies and doctored ontologies**—DICE-A, KMn, CONF etc. which have unsatisfiable classes deliberately introduced into them in order to provide further test cases due to a lack of suitable real world ontologies, or are used to investigate an optimisation along a particular dimension in a controlled way. Used in [SHCvH07, JQH09].

Prototypical Implementations The implementations used in many experiments tend to be proof of concept implementations, which are perfectly sufficient for showing the effect of some optimisation, but do not necessarily reflect the performance that could be achieved with a tuned and more robust implementation. For example, the implementation of DION was a prototypical implementation done in PROLOG. Similarly, in [SHCvH07] the Wellington reasoner [End00], which supports $\mathcal{ALC}$, was used for entailment checking in the experiments. Although Wellington features several standard optimisations, it does not appear to have been developed beyond prototype and is no longer available. It will later be seen that good entailment checking performance is crucial to the success of black-box algorithms, and use of state of the art reasoners today could paint an entirely different, more representative, picture of what can be achieved in justification finding. Similarly, in [SQJH08] the effect of modularisation as an optimisation for computing justifications was investigated with an implementation based on KAON2 [MS06]. While the implementation more than sufficed for showing how effective modularisation was as an optimisation, the use of KAON2 probably does not reflect the performance that can be obtained with modern tableau reasoners on ontologies with *rich* TBoxes. After all, KAON2 was designed as a highly optimised reasoner to be used on ontologies with very large ABoxes and simple TBoxes.

In some cases, the prototypical implementations that were used for testing were buggy. In particular, the implementation used to carry out the empirical work detailed in [Kal06], which was also used in the ontology editor Swoop, was buggy. Unfortunately, the bugs were rather critical and meant that in many cases the implementation could never compute all justifications for an entailment. The reason for this was due to an incorrectly coded termination condition in the hitting set tree construction algorithm, which caused *too* early termination. In essence, the termination condition was reversed so that paths which were subsets rather than supersets of open paths in the hitting set tree were prematurely closed.

Finally, some implementations and methods do not seem to agree on results and in some cases lead to inaccurate comparisons between techniques. For example, in [KK07] Kremen and Kouba compare an incremental glass-box approach with a pure black-box approach. However, their black-box implementation which they use for comparison takes longer than fifteen minutes to compute all justifications for all unsatisfiable classes in the miniEconomy ontology, while the black-box algorithm implementation presented in [KPHS07] can compute all justifications for all unsatisfiable classes in this ontology in under ten seconds.

In essence, most empirical work is performed on small collections of ontologies that are *chosen* to show off the effect of specific optimisations and demonstrate proof of concept. This is obviously a valid thing to do. However, many of the experiments do not provide a true picture of how highly optimised and robustly implemented justification finding techniques will perform on state of the art ontologies that are now widely available.

3.7 Summary and Directions

Overall, there is a need for empirical data that leads to well founded conclusions on how optimised techniques for computing justifications behave on realistic ontologies. While the algorithms for computing all justifications for an entailment have a high worst case complexity, little is know about how these algorithms really behave in practice. Indeed, direct criticism [Stu08] has been levelled at authors and implementors, which claims that finding justifications is not as feasible as suggested by some researchers. In particular,

> "Run-time performance is a major issue in the context of a practical application of debugging methods" [Stu08]

> "Our experiments confirmed the common knowledge that black-box approaches suffer from their computational complexity" [Stu08]

> "Not surprisingly, the two debugging tools integrated in ontology editors did not use black-box approaches. In summary, the way to move forward is to further develop [glass-box] approaches for ontology debugging..." [Stu08]

> "A high number of very expensive tableau algorithm runs required for black-box methods, as well as a lack of glass-box methods, together with their poor reusability, has given rise to the idea of using incremental techniques for axiom pinpointing" [KK07]

The issues and problems that lead to such accusations are due to the prototypical nature of proof of concept implementations and the kinds of corpora that are used for testing, and also due to the following:

Lack of Empirical Data for a Broad Corpus of Ontologies As explained above, the ontologies used in justification finding experiments in the literature have been primarily used for the purposed of testing optimisations and proof of concept. Ontologies tend to be tutorial style ontologies, auto-generated ontologies, or a narrow collection of large medical ontologies, some of which have now been significantly extended using more expressive concept and axiom constructors. These kinds of ontologies do not fully represent the expressivity and kinds of modelling style found in many published ontologies and therefore do not necessarily give the best picture of what the justification landscape might be like and what the performance of robustly implemented justification finding algorithms might be like. A large independently defined ontology corpus that represents real world state of the art ontology modelling and publishing would provide a more conclusive demonstration of how justification finding algorithms perform in practice.

Lack of Empirical Data for Inconsistent Ontologies Little is known about the practicalities of computing justifications for *inconsistent ontologies*. This is despite the fact justifications are important for repairing inconsistent ontologies, and can be used as the basis for para-consistent reasoning [Pri02]. In fact, it is easily arguable that justifications are *vital* for understanding why an ontology is inconsistent. Many tools, both reasoners and editors, simply report "inconsistent", and it is left up to the user to determine where the problem lies. However, unlike entailments in consistent ontologies, where there are structural and visual cues such as unsatisfiable classes and class names painted in red, there are no such cues that help people decide where to start searching for a problem in an inconsistent ontology.

Inconsistency can be introduced into ontologies for a number of reasons. For example, an ontology may become inconsistent after the addition of some axioms during the editing process. While the last set of axioms that were added may play a part in making the ontology inconsistent, it might actually be the case that they were the correct axioms to add from a modelling point of view, but it is the interplay of the newly added axioms with axioms already in the ontology that triggers the inconsistency. Inconsistencies can also be the result of *automated* ontology construction techniques. For example, in [DFK$^+$07], ontologies are automatically constructed as an output from text mining, and it is possible for the resulting ontologies to be inconsistent.

There are a number of different ways of dealing with an inconsistent ontology. Broadly speaking, either some form of para-consistent logic [Pri02] based reasoning can be used to draw "meaningful" conclusions from the ontology, as in [MHL07], or the ontology can be *repaired* so that it becomes consistent, thus meaningful conclusions can be drawn from the repaired ontology using standard semantics. The work presented in this thesis obviously focuses on this latter "repair strategy", with an assumed scenario that ontologies may become inconsistent during their construction process, but not wanting to publish such ontologies, their authors would repair the ontologies before publication.

Ultimately, computing justifications for inconsistent ontologies merits an investigation in its own right because inconsistent ontologies pose several problems that affect the runtime performance of justification finding algorithms. The first issue is that the notion of locality based modules break down completely when it comes to inconsistent ontologies. This means that it is not possible to compute a module for an inconsistent ontology and then perform justification finding with respect to that module, which has ramifications for entailment checking performance. The second issue, is that, without the presence of modularity entailment checking must be performed on the whole input ontology. When the input is large a selection function based incremental expansion subroutine might help improve performance. However, selection functions rely on a seed signature for selecting the initially small set of axioms. This seed signature is usually the signature of the entailment in question, but for inconsistent ontologies no such entailment or seed signature exists. This means that it is not possible to use an incremental expansion strategy to improve performance. With these challenges in mind, an investigation on the practicalities of computing justifications for inconsistent ontologies was carried out, the results of which are presented in Chapter 6.

Lack of Knowledge about Justification Metrics for Real Ontologies What is the justification landscape of state of the art ontologies like in practice? Little is known about how the size and number of justifications varies for modern ontologies. This information is important because justification algorithms exhibit worst case exponential performance in the number of justifications. However, it is not clear if there are any ontologies in the wild where the number of justifications are so high that not all justifications could be computed.

Lack of Empirical Data Showing How Glass-Box Techniques Stack Up Against Black-Box Techniques Finally, it is not particularly clear how well glass-box and black-box techniques stack up against each other. A popular belief has been that it was necessary to have explanation tightly integrated with a reasoner, i.e. glass-box based explanation, for reasons of efficiency [KK07, Stu08]—a view which goes back to before the current phase of justification based research [MBB95]. A lack of empirical data on a sizeable corpus of ontologies makes it difficult to draw watertight conclusions here.

3.7.1 Aims and Objectives

The following chapters detail a thorough investigation into computing justifications for consistent and inconsistent ontologies. The main aim is to address the points above and obtain strong evidence for the practicality of computing justifications. This is especially important as justifications continue to be used more and more as forms of explanation and as the basis for auxiliary services. The investigation also establishes a picture of the justification landscape for real ontologies and investigates how certain optimisations perform on these ontologies.

Chapter 4

Justification Finding Algorithms

This chapter presents the concrete details of the justification finding algorithms that are used in the experiments that follow in later chapters. The main purpose of this chapter is to highlight the salient features of the main algorithms, and to provide a picture of how various subroutines, such as expand-contract subroutines, fit together to form complete algorithms for finding one justification, or finding all justifications for an entailment.

4.1 Algorithm Topology

A schematic of how the various algorithms can be combined as subroutines is presented in Figure 4.1. The main routine is ComputeAllJustifications (①) which computes all justifications for an entailment η with respect to a set of axioms $\mathcal{O}$, and is listed in Algorithm 4.1. This algorithm contains the ComputeSingleJustification (②) subroutine which is used to compute single justifications for an input $\mathcal{O}$ and η, and is listed in Algorithm 4.2. In the presentation here, the ComputeSingleJustification algorithm has been generalised so that it is always regarded as being an *expand-contract* algorithm. Therefore, it always calls the ExpandAxioms (③) subroutine followed by the ContractAxioms (④) subroutine.

In the empirical investigation which follows, the ExpandAxioms (③) algorithm has three different implementations (depicted by dashed lozenges in Figure 4.1). The first is the “All in One” expansion algorithm, which is listed in Algorithm 4.3. The second is the “Incremental” expansion algorithm, which uses a selection function to gradually expand axioms based on entailment and axiom signatures and listed in Algorithm 4.5. The third is the “Tracing” expansion algorithm,

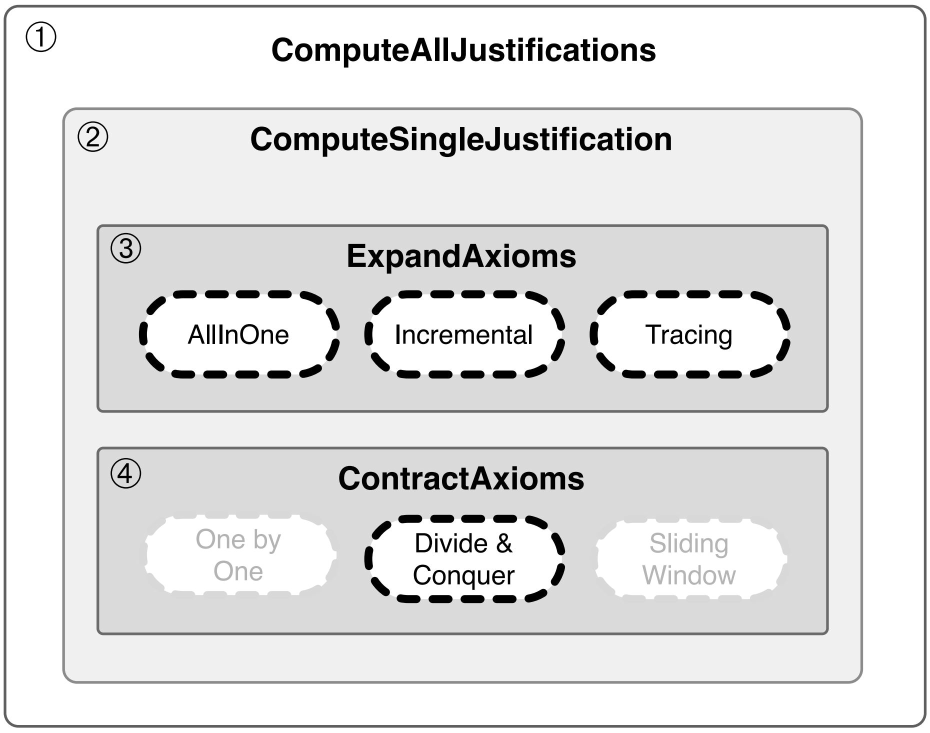


Figure 4.1: Justification Finding Algorithm Topology

which uses glass-box tracing to expand the input axioms into a justification, or slightly more than a justification, and is listed in Algorithm 4.4. In this thesis, only one contraction algorithm is investigated for the ContractAxioms subroutine, and this is the "divide and conquer" algorithm (as opposed to the "One by One" or "Sliding Window" strategies), which is listed in Algorithm 4.6. As explained in Section 3.2.2 on Page 59, there have been various experiments into the efficacy of the divide and conquer approach over other approaches such as the sliding window technique. In all cases divide and conquer contraction has been proved to offer significant performance benefits over other approaches.

4.2 Computing All Justifications for $\mathcal{O} \models \eta$

The listing shown in Algorithm 4.1 computes all justifications for an entailment $\mathcal{O} \models \eta$ using a hitting set tree T_{hst}. The tree is constructed in a breadth first manner by using a queue Q in the standard way [RN10]. The queue manipulation function $\mathsf{Enqueue}(v, Q)$ adds a node v to the tail of a queue Q, while the function $\mathsf{Dequeue}(v, Q)$ removes a node v from the head of a queue Q. There are several functions that relate to the construction and inspection of T_{hst} namely, $\mathsf{GetFreshNode}(J)$ which creates a new tree node with a given label (justification) J, $\mathsf{GetFreshEdge}(\langle v, v' \rangle, \alpha)$ which creates a new edge with a given label (axiom) α, $\mathsf{GetLabel}(v)$ which gets the label (justification) of a node v, and $\mathsf{GetPathToRootLabelSet}(v, T_{hst})$ which gets the *set* of edge labels on the path from the node v to the root of T_{hst}. Finally, the $\mathsf{ComputeSingleJustification}(S, \eta)$ returns a justification for η with respect to S if $S \models \eta$ or returns the empty set if $S \not\models \eta$.

Algorithm 4.1 Walkthrough Algorithm 4.1 computes a hitting set tree in a breadth-first manner as described in Chapter 3. It begins by extracting a module for the entailment η in $\mathcal{O}$ (line 1), which is used to initialise a set $S_{working}$ of "working axioms". In the case where $\mathcal{O}$ is inconsistent, ComputeModule is assumed to return $\mathcal{O}$ in its entirety. In lines 4–8 the algorithm computes a single justification by calling the ComputeSingleJustification sub-routine, adds this justification to the results set X_{result}, and then initialises the root of the hitting set tree T_{hst} with a node v_{root} labelled with this justification. The root node is then added to the queue Q to start the construction of the hitting set tree, which takes place in lines 9–31. The queue Q is used to construct the tree breadth first (in the usual way). A breadth first construction was chosen over a depth first construction as it is closer to Reiter's original algorithm, and allows the early path termination optimisation to be more effective. During the construction of the hitting set tree the algorithm attempts to reuse previously found justifications. It does this in line 16, where the function GetNonIntersectingJustification attempts to search for a justification that does not intersect the current path. If there are no non-intersecting justifications that can be reused, then the algorithm calls the ComputeSingleJustification subroutine to compute a fresh justification. As a further optimisation, early path termination is employed. The algorithm checks to see if the axioms that label the current path label some other path that has already been explored. If this is the case then early termination is applied to the current path.

Algorithm 4.1 Termination and Completeness Based on the fact that there is a finite number of justifications, all of finite size, it is reasonably straight forward to see that Algorithm 4.1 terminates and a proof is therefore not provided

here. Similarly, as the algorithm is fairly standard (containing standard optimisations), and constructs a hitting set which is known to contain all justifications (see [Kal06] for details), so a proof of completeness is not provided here.

Algorithm 4.1 ComputeAllJustifications($\mathcal{O}$, η)

```
1: $S_{working} \leftarrow$ ComputeModule($\mathcal{O}$, signature($\eta$))
2: $X_{explored} \leftarrow \emptyset$
3: $X_{result} \leftarrow \emptyset$
4: $J_{root} \leftarrow$ ComputeSingleJustification($S_{working}$, $\eta$)
5: $X_{result} \leftarrow X_{result} \cup \{J_{root}\}$
6: $v_{root} \leftarrow$ GetFreshNode($J_{root}$)
7: Enqueue($v_{root}$, $Q$)
8: SetRoot($T_{hst}$, $v_{root}$)
9: while $Q$ is not empty do
10:     $v_{head} \leftarrow$ Dequeue($Q$)
11:     $J_{head} \leftarrow$ GetLabel($v_{head}$)
12:     for $\alpha \in J_{head}$ do
13:         $S_{path} \leftarrow$ GetPathToRootLabelSet($v_{head}$, $T_{hst}$) $\cup \{\alpha\}$
14:         if $S_{path} \notin X_{explored}$ then
15:             $X_{explored} \leftarrow X_{explored} \cup \{S_{path}\}$
16:             $J' \leftarrow$ GetNonIntersectingJustification($S_{path}$, $X_{result}$)
17:             if $J' = \emptyset$ then
18:                 $S_{working} \leftarrow S_{working} \setminus S_{path}$
19:                 $J' \leftarrow$ ComputeSingleJustification($S_{working}$, $\eta$)
20:                 $S_{working} \leftarrow S_{working} \cup S_{path}$
21:             end if
22:             $v_{fresh} \leftarrow$ GetFreshNode($J'$)
23:             $e \leftarrow$ GetFreshEdge($\langle v_{fresh}, v_{head}\rangle, \alpha$)
24:             $T_{hst} \leftarrow T_{hst} \cup \{e\}$
25:             if $J' \neq \emptyset$ then
26:                 $X_{result} \leftarrow X_{result} \cup \{J'\}$
27:                 Enqueue($v_{fresh}$, $Q$)
28:             end if
29:         end if
30:     end for
31: end while
32: return $X_{result}$
```

4.3 Computing A Single Justification for $\mathcal{O} \models \eta$

Given $\mathcal{O} \models \eta$, Algorithm 4.2 computes a justification for η with respect to $\mathcal{O}$. The algorithm begins with a simple optimisation in lines 1–3 where it checks to see if the entailment is asserted into the input ontology $\mathcal{O}$. If this is the case, the algorithm returns the singleton set self justification. Thereafter, the algorithm simply consists of an expand-contract procedure. The actual mechanics of the expansion phase and contraction phase are delegated to the ExpandAxioms and ContractAxioms subroutines respectively.

Algorithm 4.2 ComputeSingleJustification($\mathcal{O}$, η)

```
1: if $\eta \in \mathcal{O}$ then
2:     return $\{\eta\}$
3: end if
4: $S \leftarrow$ ExpandAxioms($\mathcal{O}$, $\eta$)
5: if $S = \emptyset$ then
6:     return $\emptyset$
7: end if
8: $J \leftarrow$ ContractAxioms($S$, $\eta$)
9: return $J$
```

4.3.1 Expansion Algorithms

This thesis investigates three different expansion routines which are presented in Algorithms 4.3, 4.4 and 4.5. They are “all in one” expansion, “glass-box tracing” expansion and “incremental selection function” expansion algorithms respectively. All algorithms begin by checking that the entailment holds in the input ontology $\mathcal{O}$ which is supplied to them. If the entailment does not hold then they simply return the empty set.

All In One Expansion If the entailment holds then Algorithm 4.3, which is the “all in one” expansion routine, simply returns the complete input $\mathcal{O}$.

Tracing Expansion Algorithm 4.4, which is the “glass-box tracing” routine and is the second simplest algorithm in the presentation here, uses the initial entailment check to run a trace and obtain the axioms that support the entailment. It is assumed that the subroutine trace($\mathcal{O}, \eta$) returns these axioms, which due to

Algorithm 4.3 ExpandAxioms($\mathcal{O}$, η)

```
1: if $\mathcal{O} \not\models \eta$ then
2:     return $\emptyset$
3: else
4:     return $\mathcal{O}$
5: end if
```

the reasons explained in Section 3.4.3 on Page 69, may or may not be a justification. In Pellet 2.2.2 tracing operates during an entailment check and calling the equivalent of trace in Pellet returns the axioms used in the last check. It should therefore be noted that the initial entailment check in line 1 (which is common to all expansion algorithms) is what really computes the trace when Pellet 2.2.2 is used.

Algorithm 4.4 ExpandAxioms($\mathcal{O}$, η)

```
1: if $\mathcal{O} \not\models \eta$ then
2:     return $\emptyset$
3: else
4:     $S \leftarrow$ trace($\mathcal{O}$, $\eta$)
5:     return $S$
6: end if
```

Incremental Expansion Algorithm 4.5 is the most involved of all algorithms as it uses a selection function to perform incremental expansion on axioms in the input $\mathcal{O}$.

As mentioned, the incremental expansion of axioms takes place in the expansion loop in Algorithm 4.5. The basic strategy is to take into consideration the signature of the entailment and the signature of any already expanded axioms, and then to increase the set of expanded axioms by ensuring that all defining terms for the signature are present in the expansion set S. The notion of defining axioms is made clear in Table 4.1, where for a concept, role or individual name the forms of defining axioms are shown in the last column. The selection function has one quirk which is not shown explicitly in Algorithm 4.5 but relates to how the signature is extracted from a set of axioms (line 13 in Algorithm 4.5). The GetSignature subroutine in line 13 does not include the signature of disjoint concept axioms in the returned set of names. The reason for this is that the Web Ontology Language, OWL, features disjoint concept axioms and in

Algorithm 4.5 ExpandAxioms($\mathcal{O}$, η)

```
1:  if $\mathcal{O} \not\models \eta$ then
2:      return $\emptyset$
3:  else
4:      $S \leftarrow \emptyset$
5:      $S' \leftarrow \emptyset$
6:      $\Sigma \leftarrow \mathsf{signature}(\eta)$
7:      repeat
8:          $S' \leftarrow S$
9:          $S \leftarrow S \cup \mathsf{GetDefiningAxioms}(\Sigma, \mathcal{O})$
10:         if $S \models \eta$ then
11:             return $S$
12:         end if
13:         $\Sigma \leftarrow \mathsf{GetSignature}(S)$
14:     until $S' = S$
15:     return $\mathcal{O}$
16: end if
```

some camps the practice of using these axioms to make asserted sibling classes pairwise disjoint [RDH+04] is advocated as part of the process of "ontology normalisation" [Rec03]. As can be imagined, this can result in a huge blowup in the number of axioms that are selected based on usage and shared signature. The pairwise disjoint axioms pull in all sibling classes and, after two steps, pull in all definitions for the sibling classes. Treating disjoint concept axioms in this special way helps to ameliorate this situation, and while the effect of this optimisation is not explicitly investigated in this thesis, casual experiments have shown that it produces a positive effect in ontologies where this modelling style is present.

Soundness and Completeness of Expansion Algorithms The soundness and completeness of the above expansions algorithms is obvious. For a given input $\mathcal{O}$ and η all of the algorithms return a non-empty set of axioms $\mathcal{S} \subseteq \mathcal{O}$ such that $\mathcal{S} \models \eta$ if and only if $\mathcal{O} \models \eta$ and otherwise return the empty set. All of the algorithms feature an initial entailment check to see if $\mathcal{O} \models \eta$. If the entailment does not hold then the empty set is returned. If the entailment does hold then either some subset of $\mathcal{O}$ that entails η is returned (Algorithm 4.4 and Algorithm 4.5) or the input $\mathcal{O}$ is returned (Algorithm 4.3, and Algorithm 4.5) if no proper subset of $\mathcal{O}$ that entails η is found.

Table 4.1: Incremental Expansion Function Defining Axioms. A is a concept name, C is a (possibly complex) concept, R is a role name, S is a (possibly inverse) role, and a and b are an individual names.

Entity	**Type**	**Defining Axiom Forms**
A	Concept name	$A \sqsubseteq C$
		$C \sqsubseteq A$
		$A \equiv C$
R	Role name	$R \sqsubseteq S$
		$R \equiv S$
		$\mathsf{trans}(R)$
a	Individual name	$C(a)$
		$R(a, b)$

4.3.2 A Divide and Conquer Contraction Algorithm

Algorithm 4.6 ContractAxioms($\mathcal{O}$, η)

```
  return ContractAxiomsRecursive($\emptyset$, $\mathcal{O}$, $\eta$)
ContractAxiomsRecursive($S_{support}$, $S_{whole}$, $\eta$)
 1: if $|S_{whole}| = 1$ then
 2:     return $S_{whole}$
 3: end if
 4: $S_L, S_R \leftarrow$ Split($S_{whole}$)
 5: if $S_{support} \cup S_L \models \eta$ then
 6:     return ContractAxiomsRecursive($S_{support}$, $S_L$, $\eta$)
 7: end if
 8: if $S_{support} \cup S_R \models \eta$ then
 9:     return ContractAxiomsRecursive($S_{support}$, $S_R$, $\eta$)
10: end if
11: $S'_L \leftarrow$ ContractAxiomsRecursive($S_{support} \cup S_R, S_L, \eta$)
12: $S'_R \leftarrow$ ContractAxiomsRecursive($S_{support} \cup S'_L, S_R, \eta$)
13: return $S'_L \cup S'_R$
```

For previously explained reasons the only contraction strategy that is considered in this thesis is a classic divide and conquer strategy. A recursive divide and conquer contraction algorithm is presented in Algorithm 4.6. The algorithm takes as an input a set of support axioms $S_{support}$, the set of axioms upon which a binary chop will be performed S_{whole}, and the entailment η (where $(S_{support} \cup S_{whole}) \models \eta$).

First, the base case where S_{whole} contains one axiom is checked. Next, the algorithm splits S_{whole} into two halves using the Split subroutine. Each half is checked to see if it independently entails with the set of support axioms. If one of the halves entails η then the other half is essentially thrown away (lines 5–10). If justifications for η are spread across both halves then each half is divided and search by a recursive call of the routine (lines 11–12) and the results are combined and returned (line 13).

4.4 Summary

This chapter has presented the algorithms that are used for computing justifications. In the next chapter an ontology corpus is described which the algorithms are then evaluated on.

Chapter 5

The BioPortal Corpus

In the next chapter an empirical evaluation of the previously presented algorithms is described. This evaluation was performed on ontologies contained in the NCBO BioPortal ontology repository [DLR08, NDG$^+$09, NSW$^+$09]. This chapter describes the BioPortal, why it makes a good corpus, and how the ontologies in it were processed and whittled down to the particular corpus used in the empirical evaluation.

5.1 Ontology Corpus

The number of published real world ontologies has grown significantly since computing justifications for entailments in OWL ontologies was first investigated from 2003 onwards. In particular, the number of ontologies in the biomedical arena has grown considerably. Many of these ontologies have been made available via the *NCBO BioPortal* ontology repository [DLR08, NDG$^+$09, NSW$^+$09], which is used for accessing and sharing "ontologies that are actively used in biomedical communities" [DLR08]. At the time of writing, BioPortal provides access to the imports closures of over 250 bio-medical ontologies in various formats, including OWL and OBO [SAR$^+$07] ontologies.

Not only is BioPortal useful for end users who want to share and use biomedical ontologies, it is also useful for ontology tools developers as it provides a corpus of ontologies that is attractive for the purposes of implementation testing. The BioPortal corpus is attractive for several reasons; in particular, it provides ontologies that:

- Vary greatly in size. Ontologies range from hundreds of axioms in size through to hundreds of thousands of axioms in size.

- Vary greatly in expressivity. The expressivity of the Description Logics used to represent BioPortal ontologies ranges from light weight $\mathcal{EL}$ through to highly expressive $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$.

- Are real world ontologies. The ontologies in BioPortal are not tutorial or reasoner test bed ontologies. They were designed and built by users (domain experts) for application purposes. In this sense, they provide a window on the state of the art in ontology engineering.

- Are developed by a wide range of groups and developers. They contain a variety of modelling styles from simple taxonomies through to axioms containing deeply nested complex concepts, and therefore offer a good perspective on the kinds of ontologies that people build.

- Are not "cherry picked" to show good performance of tools. Any biomedical ontology developer can add their ontology to BioPortal.

One natural concern with a corpus based on BioPortal might be that all of the ontologies come from the same domain. Specifically, it could be argued that this restricts the pool of ontologies to ones with a distinctive modelling style, that arises due to the subject matter being modelled, and widespread bio-ontology community modelling principles and diktats. Indeed, there is a perception that many bio-medical ontologies only require lightweight description logics, such as $\mathcal{EL}$, and are heavily influenced by modelling guidelines such as those dispensed by the OBO Foundry [SAR+07]. However, as explained above, the BioPortal corpus contains a wide variety of expressive and logically rich ontologies that belie these perceptions. Moreover, while the BioPortal does contain some ontologies that have been contributed by several well known communities, many of the ontologies have been submitted by a wide range of authors that come from independent communities, and eyeballing these ontologies reveals different modelling styles are present throughout the corpus. Finally, to date, the BioPortal corpus provides the most thorough test-bed for justification finding services, and far surpasses any other collection of ontologies used for testing in this area. In essence, this point, in combination with the itemised points above, must outweigh any concerns about domain bias.

5.1.1 Curation Procedure

The BioPortal ontology repository was accessed on the 12th March 2011 using the BioPortal RESTful Service API. In total, 261 ontology documents (and their imports closures) were listed as being available. Out of these, there were 125 OWL ontology documents, and 101 OBO ontology documents, giving a total of 226 "OWL compatible" ontology documents that could theoretically be parsed into OWL ontologies.

Parsing and Checking

Each listed OWL compatible ontology document was downloaded and parsed by the OWL API. OBO ontology documents were parsed according to the BNF given by Horrocks and Golbreich in [GH07] described further in [GHH$^+$07] and extended by Mungall [Mun11]. Any imports statements were recursively dealt with by downloading the document at the imports statement URL and parsing it into the imports closure of the original BioPortal "root" ontology. Each axiom that was parsed into the imports closure was labelled with the name of the ontology document from where it originated. If an imported ontology document could not be accessed (for whatever reason) the import was silently ignored.

Out of the 226 OWL compatible ontology documents that were listed by the BioPortal API, 7 could not be downloaded due to HTTP 500[1] errors, and 1 ontology could not be parsed due to syntax errors. This left a total of 218 OWL and OBO ontology documents that could be downloaded parsed into OWL ontologies. After parsing, four of the ontologies were found to violate the OWL 2 global restrictions. In all cases, the violation was caused by the use of transitive (non-simple) properties in cardinality restrictions. These ontologies were discarded and were not processed any further, which left 214 ontologies.

Pruning

As explained in the preliminaries chapter, this thesis concentrates on the Description Logic $\mathcal{SHOIQ}$. There were five ontologies that contained a mixture of reflexive role axioms, irreflexive role axioms, and role chain subsumption axioms, which lie outside the expressivity of $\mathcal{SHOIQ}$. Rather than completely discarding these

[1]An HTTP 500 error is an error code that indicated the web server encountered an internal error that prevented it from fulfilling the client request.

ontologies the non-$\mathcal{SHOIQ}$ axioms were removed from them as follows: Ontology 11 (cancer-research-and-management-acgt-master-ontology)—1 irreflexive property axiom removed. Ontology 47 (oboe)—2 role chain axioms removed. Ontology 48 (oboe-sbc)—2 role chain axioms removed. Ontology 61 (semanticscience-integrated-ontology)—3 reflexive role axioms removed, 3 irreflexive role axioms removed, 1 role chain axiom removed. Ontology 70 (uber-anatomy-ontology)—1 reflexive role axiom removed. The decision to repair and retain these ontologies, as opposed to completely discarding them, was taken because the removal of non-$\mathcal{SHOIQ}$ axioms from them was entirely deterministic. That is, an axiom was removed if and only if it is a non-$\mathcal{SHOIQ}$ axiom—it was not necessary to choose one repair/conversion plan over another and make an arbitrary choice that could have affected the results of the experiments that follow in an arbitrary way. It should be noted that this is in contrast to the repair of an ontology that violates the OWL 2 global restrictions, where there could be multiple reasons, and therefore multiple alternative repair plans for that ontology.

Entailment Extraction

Three reasoners were used for entailment extraction: HermiT, JFaCT, which is a Java port of FaCT++[2], and Pellet. Each ontology was checked for consistency. Five of the 214 ontologies were found to be inconsistent. Next, each *consistent* ontology was classified and realised in order to extract entailments to be used in the justification finding experiments. Entailed *direct* subsumptions between named classes (i.e. axioms of the form $A \sqsubseteq B$) were extracted, along with *direct* class assertions between named individuals and named classes (i.e. axioms of the form $A(a)$). It was decided that these kinds of entailments should be used for testing purposes because they are the kinds of entailments that are exposed through the user interfaces of tools such as Protégé-4 and other ontology browsers, and are therefore the kinds of entailments that users of these tools typically seek justifications for. The set of entailments for each ontology was then filtered so that it only contained *non-trivial* entailments in accordance with Definition 5.

[2]JFaCT was used in place of FaCT++ because it was found that JFaCT could handle more BioPortal ontologies than FaCT++. This was due to the reason that, at the time of writing, JFaCT supports more datatypes than FaCT++ does and the BioPortal corpus contains several ontologies whose signatures contain datatypes that cannot be handled by FaCT++. It is worth noting that JFaCT and FaCT++ are comparable in terms of performance.

Definition 5 (Non-Trivial Entailment)**.** *Given an ontology $\mathcal{O}$, such that $\mathcal{O} \models \alpha$, the entailment α in $\mathcal{O}$ is non-trivial if $\mathcal{O} \setminus \{\alpha\} \models \alpha$*

Intuitively, for an ontology $\mathcal{O}$ and an entailment α such that $\mathcal{O} \models \alpha$, α is a non-trivial entailment in $\mathcal{O}$ either if α is not asserted in $\mathcal{O}$ (i.e. $\alpha \notin \mathcal{O}$) or, α is asserted in $\mathcal{O}$ (i.e. $\alpha \in \mathcal{O}$) but $\mathcal{O} \setminus \{\alpha\} \models \alpha$, i.e. $\mathcal{O}$ with α removed still entails α. In total there were 72 ontologies with non-trivial entailments which accounts for just over one third of the consistent OWL and OBO ontologies contained in BioPortal.

Reasoner Performance

Due to practical considerations, a timeout of 30 minutes of CPU time was set for each task of consistency checking, classification and realisation. There were just three ontologies, for which consistency checking (and hence classification and realisation) could not be completed within this time out. These were: GALEN, the Foundational Model of Anatomy (FMA) [RJ03], and NCBI Organismal Classification. The version of GALEN contained within the BioPortal is one of the extremely hard versions of the ontology that only the CB reasoner [Kaz09] is able to deal with. Since CB is written in objective-camel and does not integrated with the OWL API it was not used for testing. It is worth noting that, in the literature there are various published reasoning service experiments which successfully use GALEN as a test case. However, these experiments typically use doctored versions of GALEN, which are easy for reasoners such as FaCT++, Pellet and HermiT to perform consistency checking on, and accounts for differences between third party published results and the results obtained here.

5.1.2 Ontologies With Non-Trivial Entailments

The set of BioPortal ontologies that contained at least one non-trivial entailment is shown in Table 5.1. For these ontologies, the average number of logical axioms (i.e. non-annotation axioms) per ontology was 10,645 (SD=31,333, Min=13, Max=176,113). The average number of non-trivial entailments per ontology was 1,548 (SD=6,187, Min=1, Max=49,537). In terms of expressivity, as can be seen, the expressivity of the BioPortal ontologies with non-trivial entailments ranges from $\mathcal{EL}$ and $\mathcal{EL}++$ through to $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$, with various levels of expressivity in between these two ends of the scale.

5.2 Summary

In summary, the ontology corpus provided by the BioPortal exhibits varying numbers of non-trivial entailments with a wide range of expressivities. It reflects current modelling practices and the kinds of ontologies that people used in tools.

Table 5.1: BioPortal Corpus Ontologies

Id	**Ontology**	**Expressivity**	**Axioms**	**Entailments**	**Non-Triv Ents.**
1	adverse-event-ontology	$\mathcal{SHI}$	548	426	49
2	adverse-event-reporting-ontology	$\mathcal{SHOIN}$	513	256	91
3	amino-acid	$\mathcal{ALCF}$	497	49	44
4	basic-formal-ontology	$\mathcal{ALC}$	95	38	35
5	basic-vertebrate-anatomy	$\mathcal{SHIF}$	388	116	33
6	biological-imaging-methods	$\mathcal{EL}++$	536	519	43
7	biopax	$\mathcal{SHIN}$	391	68	4
8	biotop	$\mathcal{SRI}$	680	357	357
9	bleeding-history-phenotype	$\mathcal{ALCIF}$	1925	615	87
10	brucellosis-ontology	$\mathcal{SHOI}$	1279	889	283
11	cancer-research-and-management-acgt-master-ontology	$\mathcal{SROIQ}$	5522	1808	225
12	cardiac-electrophysiology-ontology	$\mathcal{SHF}$	176113	81689	24
13	cell-behavior-ontology	$\mathcal{ALUO}$	13	11	7
14	cell-type	$\mathcal{ALC}$	2301	1864	18
15	cereal-plant-gross-anatomy	$\mathcal{EL}++$	1868	1090	2
16	chemical-information-ontology	$\mathcal{SHIN}$	752	7346	7016
17	cognitive-paradigm-ontology	$\mathcal{SHOIN}$	651	370	67
18	computer-based-patient-record-ontology	$\mathcal{SHOIN}$	1480	472	348
19	coriell-cell-line-ontology	$\mathcal{ALCHI}$	139004	51375	49537
20	dendritic-cell	$\mathcal{ALC}$	313	145	47
21	dermlex-the-dermatology-lexicon	$\mathcal{ALUF}$	24452	12204	6099
22	electrocardiography-ontology	$\mathcal{ALCIF}$	1028	866	47
23	evidence-codes	$\mathcal{EL}++$	321	315	112
24	experimental-factor-ontology	$\mathcal{SHOIF}$	7094	4835	2000

Id	**Ontology**	**Expressivity**	**Axioms**	**Entailments**	**Non-Triv Ents.**
25	family-health-history-ontology	$\mathcal{ALCHIF}$	1103	325	144
26	gene-ontology-extension	$\mathcal{EL}++$	60293	45518	8082
27	gene-regulation-ontology	$\mathcal{ALCHIQ}$	962	563	100
28	general-formal-ontology	$\mathcal{SHIQ}$	212	45	10
29	geospecies-ontology	$\mathcal{SHOIN}$	3447	786	8
30	host-pathogen-interactions-ontology	$\mathcal{SHI}$	403	274	35
31	human-disease	$\mathcal{EL}++$	10281	10307	47
32	imgt-ontology	$\mathcal{ALCIN}$	1112	108	35
33	infectious-disease	$\mathcal{SHOI}$	633	556	56
34	information-artifact-ontology	$\mathcal{SHOIN}$	312	149	68
35	interaction-network-ontology	$\mathcal{ALC}$	1034	977	35
36	international-classification-for-nursing-practice	$\mathcal{SHIF}$	12958	3860	2230
37	international-classification-of-external-causes-of-injuries	$\mathcal{AL}$	13612	4458	2225
38	international-classification-of-functioning-disability-and-health-icf-	$\mathcal{ALUHIF}$	13913	3930	2411
39	linkingkin2pep	$\mathcal{SHIF}$	30	7	2
40	lipid-ontology	$\mathcal{ALCHIN}$	2375	719	275
41	mahco-an-mhc-ontology	$\mathcal{ALCIQ}$	13844	10291	26
42	mged-ontology	$\mathcal{ALEOF}$	2545	1003	2
43	nci-thesaurus	$\mathcal{SH}$	146488	89468	16202
44	neomark-oral-cancer-centred-ontology	$\mathcal{ALCHQ}$	1755	62	48
45	neural-electromagnetic-ontologies	$\mathcal{SHIQ}$	2405	1513	148
46	nmr-instrument-specific-component-of-metabolomics-investigations	$\mathcal{SH}$	599	477	36
47	oboe	$\mathcal{SRIQ}$	265	29	18
48	oboe-sbc	$\mathcal{SRIQ}$	1096	459	108
49	ontology-for-biomedical-investigations	$\mathcal{SHOIN}$	25272	3329	703
50	ontology-for-drug-discovery-investigations	$\mathcal{SHOIN}$	968	666	62

Id	Ontology	Expressivity	Axioms	Entailments	Non-Triv Ents.
51	ontology-for-general-medical-science	$\mathcal{ALCO}$	213	147	54
52	ontology-for-genetic-interval	$\mathcal{SHIN}$	509	223	51
53	ontology-of-clinical-research-ocre-	$\mathcal{ALCHOIN}$	971	365	99
54	ontology-of-glucose-metabolism-disorder	$\mathcal{ALC}$	3195	175	35
55	ontology-of-medically-related-social-entities	$\mathcal{ALCO}$	155	91	52
56	phare	$\mathcal{ALCHIF}$	459	228	34
57	pilot-ontology	$\mathcal{ALCIF}$	85	20	1
58	plant-ontology	$\mathcal{EL}++$	2107	1325	2
59	protein-ontology	$\mathcal{ALCF}$	691	68	28
60	protein-ontology-pro-	$\mathcal{S}$	26854	26089	4682
61	semanticscience-integrated-ontology	$\mathcal{SRIQ}$	1492	922	55
62	sequence-types-and-features	$\mathcal{SHI}$	2398	1830	244
63	skin-physiology-ontology	$\mathcal{ALERIF}+$	678	357	40
64	snp-ontology	$\mathcal{SHOIN}$	10373	2409	566
65	software-ontology	$\mathcal{ALCHIN}$	2080	727	332
66	spatial-ontology	$\mathcal{ALEHI}+$	234	165	120
67	subcellular-anatomy-ontology-sao-	$\mathcal{SHIN}$	2935	820	59
68	terminology-for-the-description-of-dynamics	$\mathcal{ALCRIQ}$	286	176	51
69	translational-medicine-ontology	$\mathcal{SHIN}$	380	216	46
70	uber-anatomy-ontology	$\mathcal{EL}++$	15939	8075	3997
71	vaccine-ontology	$\mathcal{SHOIN}$	8444	4453	1293
72	vertebrate-anatomy-ontology	$\mathcal{EL}++$	307	188	6

Chapter 6

Justification Finding Experiments

This chapter presents several experiments that were carried out on the BioPortal corpus of ontologies. As stated previously, the main purpose of the experiments was to investigate the practicability of computing justifications for entailments in real world ontologies. The first experiment uses an implementation of Algorithm 4.1 to compute all justifications for direct atomic subsumptions and class assertions. The data obtained from this experiment is used to draw conclusions about the practicality of computing all justifications using model based diagnosis techniques, as well as providing an insight into the justification landscape of the set of BioPortal ontologies. Next, several experiments which investigate the effects of selective expansion and glass-box expansion are presented. These experiments are performed on random samples of entailments from the BioPortal corpus. Finally, a sample of inconsistent ontologies that were found on the world wide web and various mailing lists are used to examine the practicality of computing justifications for inconsistent ontologies.

Choice of Reasoner All of the experiments were carried out using Pellet 2.2.2 backed by the OWL API [BVL03, HB09, HB11]. This API has support for manipulating ontologies at the level of axioms, and so it is entirely suited for the implementation of the justification finding algorithms that have been described previously. The decision to focus on this reasoner was taken for the following reasons:

1. **Tableau tracing**—The primary reason for using Pellet is that it is the

only mainstream reasoner which supports some form of tableau tracing. This provides a level of uniformity across the entire set of experiments that follow. In particular it makes it easy to compare black-box and glass-box approaches.

2. **Robustness of Dealing with BioPortal Ontologies**—In the BioPortal ontology processing and entailment extraction experiments described in the previous chapter it was found that Pellet was the most robust reasoner in terms of being able to deal with input syntax (in particular dealing with with certain xsd:datatypes) perform consistency checking, classification and entailment extraction on the BioPortal ontologies. Indeed, although HermiT and JFaCT performed very well, it was Pellet which had the fewest number of failures for the combination of syntax and reasoner tasks.

3. **Robustness and Conformance with the OWL API**—Pellet conforms rather well to the OWL API specification of reasoner interfaces. In particular, it provides robust and reliable support for setting timeouts for entailment checking.

Finally, in addition to the above choice criteria, all three reasoners were more or less "in the same ball-park" in terms of consistency checking, classification and entailment extraction. It is therefore reasonably safe to assume that the results which follow can be extrapolated to other reasoners such as HermiT, JFaCT, FaCT++ and Racer.

Percentile Plots Many of the results that follow are presented as percentile plots. These kinds of plots tend to illustrate the variation in some variable e.g. time, better than a plot of the mean of that variable alone. This is because they expose the extremes as well as painting a picture of the middle ground, and therefore give some feeling for typical and worst case performance. In order to understand how percentiles work, consider the problem of computing all justifications for a set of entailments in an ontology. The nth percentile for this problem represents the time taken for some entailment in that set such that n percent of entailments took the same time or less. The 50th percentile corresponds to the median. When the number and size of justifications per entailment is displayed, the ordering is *reversed*, so for example, the nth percentile represents the number of justifications for some entailment such that n percent of entailments in the set had same or *more* justifications.

Having introduced the overall test setup, each experiment is now described in detail.

6.1 Finding All Justifications

Experiment 1 was carried out in order to determine the practicability of computing all justifications for an entailment. The experiment primarily focuses on the use of a hitting set tree for computing justifications and uses a black-box implementation of the sub-routine for finding single justifications. As will be seen later, the behaviour of glass-box find-one algorithms should allow the results to be generalised to other find-one implementations.

Experiment 1: Black Box Find All

Experiment 1 Algorithm Implementation

Algorithm 4.1 and its subroutines: Algorithm 4.2, Algorithm 4.5 and Algorithm 4.6 were implemented in Java against the OWL API version 3.2.2.

Experiment 1 Test Data

The test data consisted of the 72 BioPortal ontologies that contained non-trivial entailments. For each ontology, the set of *all* non-trivial direct subconcept ($A \sqsubseteq B$) and direct concept assertion ($A(a)$) entailments were extracted and paired up with the ontology.

Experiment 1 Method

The experiments were performed on MacBook Pro with a 3.06 GHz Intel Core 2 Duo Processor. The Java Virtual Machine was allocated a maximum of 4 GB of RAM. Pellet 2.2.2 was used as a backing reasoner for performing entailment checks, with each entailment check consisting of a load, followed by a query to ask whether or not the entailment held. The implementation of Algorithm 4.1 was used to compute all justifications for each non-trivial entailment for each

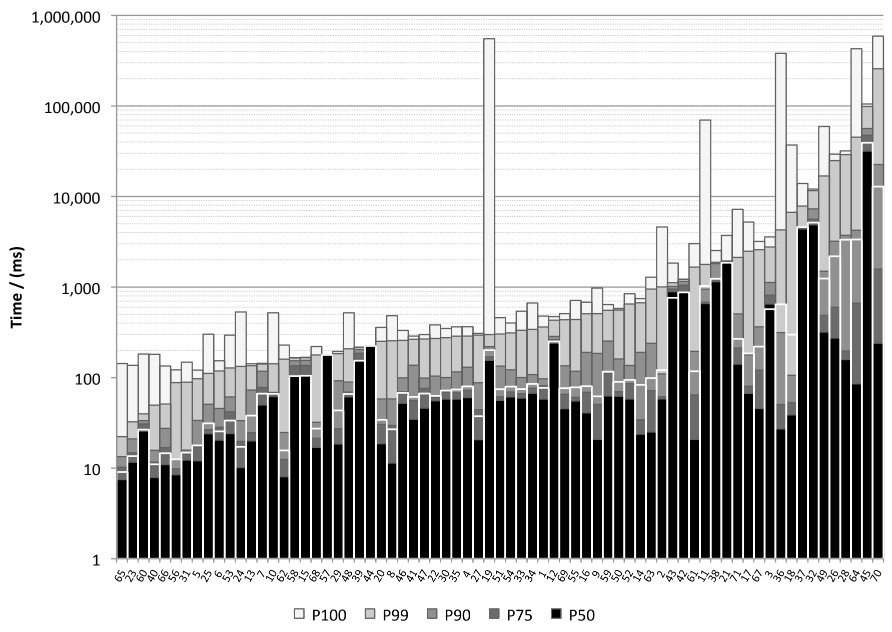


Figure 6.1: Percentile and Mean Times to Compute All Justifications Using Algorithm 4.1 with Algorithm 4.2 (Expand-Contract Find One), Algorithm 4.5 (Expansion by Selection Function) and Algorithm 4.6 (Divide and Conquer) as sub-routines. Mean times are shown in white outlines. The x-axis (Ontology) is sorted by the 99th percentile (P99).

ontology. For each entailment, the CPU time for computing all justifications was measured, along with the number and sizes of justifications. For the sake of practicalities, because some ontologies have huge numbers of non-trivial entailments (e.g. 49,000+ entailments for the coriell-cell-line ontology), a soft time limit of 10 minutes was imposed on computing all justifications for any one entailment. Additionally, an entailment test time limit of 5 minutes was placed on entailment checking.

Experiment 1 Results

Figure 6.1 shows a percentile plot for time to compute all justifications. Note that mean values for each ontology are shown as transparent bars with white outlines. The x-axis, which shows Ontology Id, is ordered by the value of the 99th percentile. This percentile was chosen because it provides a good picture of how the algorithm will perform in practice for the vast majority of entailments. It also draws out the remaining 1 percent of outliers rather clearly.

Table 6.1: Black-Box Find All Timeouts

Ont.	**Ents.**	**Failed**	**% Failed**	**Computed Justs.**				**Entailment Check**		
				Number		**Size**		**Time / (ms)**		
				Mean	**Max**	**Mean**	**Max**	**Mean**	**SD**	**Max**
19	49537	16	0.03	40.6	65	15.7	23	1.1	0.4	3
36	2230	1	0.04	414.0	414	13.5	17	0.5	0.3	2
64	566	4	0.71	1284.5	1411	25.1	28	2.5	1.6	13
70	3997	188	4.70	448.5	1060	12.7	24	7.0	77.8	141,721
45	148	9	6.08	1.6	2	21.9	24	1,979.9	6,762.4	41,840
3	44	3	6.82	1271.3	1494	26.9	35	2.5	1.3	10
32	35	26	74.29	2.2	7	2.5	8	2,601.2	23,781.0	270,004

There were *seven* ontologies that contained one or more entailments for which it was *not* possible to compute *all* justifications. These ontologies, along with the total number of entailments and the number of failed entailments are shown in Table 6.1. Table 6.1 also shows the mean/max number and size of justifications per failed entailment and the mean entailment checking times per failed entailment. In these seven ontologies there were three ontologies for which the failures occurred over less than one percent of entailments tested, a further ontologies where the failures occurred for less than 7 percent of entailments tested, and one final ontology, where failures occurred for almost 75 percent of entailments tested. In this last ontology *all* of the failures were due to entailment checking timeouts. The failures relating to all of the other ontologies were due to timeouts during construction of the hitting set tree, which became too large to search within a period of 10 minutes.

Figure 6.2 and Figure 6.3 provide a picture of the justification landscape for the BioPortal ontologies. Figure 6.2 shows the the mean number of justifications per entailment per percentile. It should be noted that the percentiles are calculated from a *reverse* ordering of entailments based on justification size. That is, the nth percentile contains n percent of entailments that have the *largest* number of justifications. The x-axis in Figure 6.2 is ordered by the mean value of the 100th percentile (i.e. mean number of justifications per entailment). Figure 6.3 shows the mean number of axioms per justification per percentile along with the maximum number of axioms per entailment. The percentiles are calculated from a reverse ordering on justification size. For example, the nth percentile contains n percent of justifications that have a mean size *greater* than the mean of that percentile.

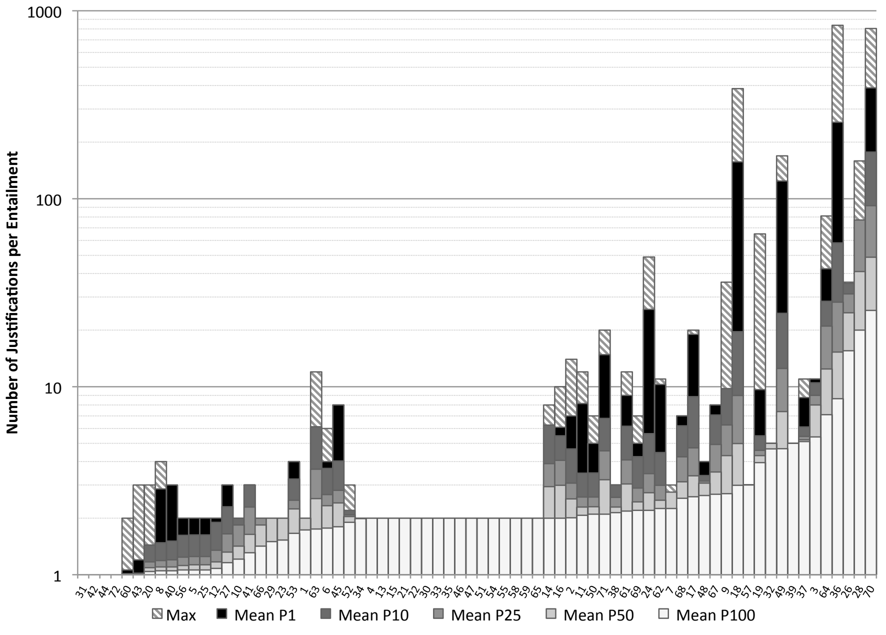


Figure 6.2: The Mean Number of Justifications per Entailment for Various Percentiles (Percentiles of the Entailments Sorted by the Number of Justifications in Descending Order—e.g. P10 represents the top 10 percent of entailments with the highest number of justifications per entailment.)

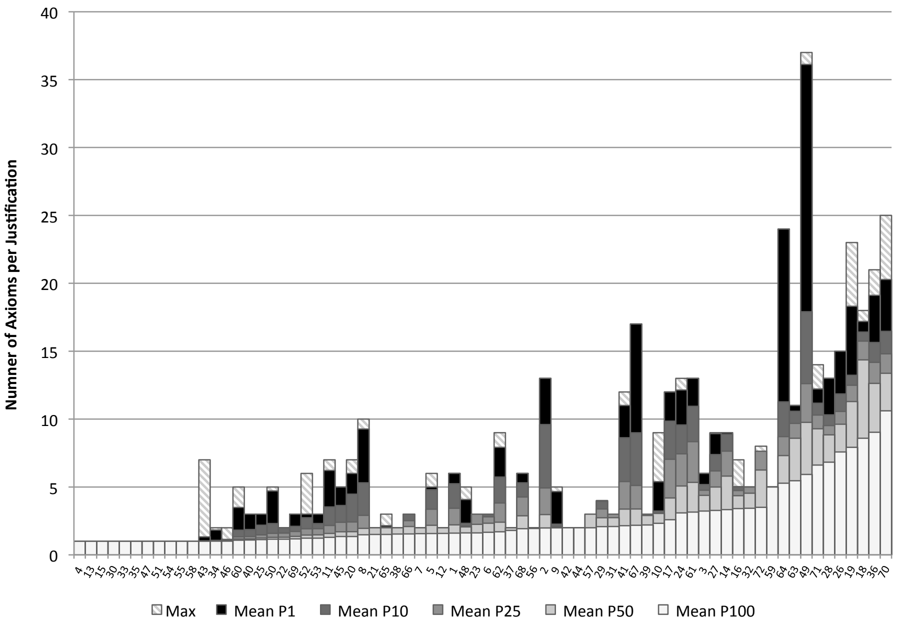


Figure 6.3: Mean Number of Axioms per Justification (Percentiles of Justifications Sorted by the Size of Justifications in Descending Order—e.g. P10 represents the top 10 percent of justifications with the largest size.)

Experiment 1 Analysis

The Practicalities of Computing All Justifications for An Entailment Out of the 72 ontologies it was possible to compute all justifications for all direct atomic subconcept and concept assertion entailments in 65 ontologies. There were seven ontologies that contained some entailments for which not all justifications could be computed. These failures are discussed below, however, the results from this experiment provide strong empirical evidence that it is largely practical to compute all justifications for these kinds of entailments in the BioPortal ontologies. Although the results cannot be statistically generalised to ontologies outside of the BioPortal corpus it is reasonable to assume that the results are suggestive for other real world ontologies.

Reasons for Failures Seven of the 72 ontologies contained entailments for which not all justifications could be computed. Broadly speaking there were two reasons for this:

1. The justifications for each failed entailment were numerous and large in size. This resulted in the size of the hitting set tree growing to a limit where it was not possible to close all branches within 10 minutes. In particular, for Ontology 36 the hitting set tree grew to over 3 million nodes, and for Ontology 70 the hitting set tree grew to over 1.6 million nodes. This compares to hitting set tree sizes in the tens of thousands for successful entailments.

2. Entailment checking performance was such that the number of entailment checks, in combination with the time for each check, made it impossible to construct the hitting set tree within 10 minutes. This was the case with Ontology 45 and Ontology 32, both of which had average entailment checking times that were three orders of magnitude higher than for other ontologies. This problem was particularly endemic for Ontology 32, which suffered the largest number of failures, and had the worst entailment checking performance of all ontologies (M=2,601.2 ms, SD=23,781.0 ms, MAX=270,004 ms)

Leaving aside entailment checking performance problems, which can be regarded as being out of the scope of control of this work, the number of justifications that were computable for failed entailments was very high. For example, for Ontology 3, which percentage-wise suffered the highest number of failures, the implementation was still able to compute on average 1271 justifications per failed entailment, with a maximum of 1494 justifications. With the exception of Ontology 32, which had a very high percentage of failures, it is fair to say that over the whole corpus, and within individual ontologies, the failure rate is low to very low, thus indicating the robustness of the algorithms on real world ontologies. When the algorithm does fail to find all justifications the number of found justifications tends to be very large.

The Acceptability of Times for Computing All Justifications As can be seen from Figure 6.1, the majority of ontologies contained entailments for which all justifications could be computed within 1 second. For all but six ontologies, the implementation of Algorithm 4.1 was able to compute all justifications for 99 percent of entailments within 10 seconds. Only two ontologies required longer than one minute for computing all justifications for 99 percent of entailments in these ontologies, with 90 percent of entailments in these ontologies falling below the one minute mark. It is clear to see that there are some outlying entailments in the corpus. In particular, Ontology 19 (the Coriell Cell Line Ontology) contains the most significant outlier, with one percent of entailments in this ontology requiring almost 150 seconds for computing all justifications. However, it appears that the times are perfectly acceptable for the purposes of generating justifications for debugging or repair in ontology development environments.

The Number of Justifications per Entailment As can be seen from Figure 6.2, the number of justifications per entailment varied over much of the BioPortal corpus. There were just four ontologies which had on average one justification per entailment. Even ontologies with low average numbers of justification per entailment did exhibit some entailments with large numbers of justifications as evidenced by the band of ontologies from 60 to 52 on the left hand side of Figure 6.2. On the right hand side of Figure 6.2, the band of ontologies from 14 through to 70 represent ontologies with very large numbers of justifications per entailment. For example, Ontology 70 had on average 25 justifications per entailment, with 10 percent of entailments having over 177 justifications, and 50 percent of entailments having over 48 justifications. This was closely followed by Ontology 28, which had on average 20 justifications per entailment, with 50 percent of entailments having over 40 justifications. The maximum number of justifications for any one entailment occurred in Ontology 36, which had 837 justifications for one entailment. At this point it is worth noting that *none* of the empirical work detailed in the literature uncovered ontologies with these large and very large numbers of justifications per entailment.

The Size of Justifications Figure 6.3 shows the mean numbers of axioms per justification per ontology. The ontologies are ordered by mean number of axioms per justification. There is a clear band of ontologies to the left hand side of Figure 6.3 that only have, on average, one axiom per justification. Recall that each entailment is a non-trivial entailment, which means that these justifications are not simply "self" justifications. In general the mean values (100th percentile) for each ontology are fairly low, with only 11 ontologies (shown on the right hand side of Figure 6.3) having over 5 axioms per justification on average . However, as witnessed by the 1st, 10th, 25th and 50th percentile columns in Figure 6.3, there are in fact many ontologies with many entailments that have larger numbers of axioms per justification. For example there are 10 ontologies where 50 percent of justifications contained over 7 axioms, and 10 percent of justifications contained 10 to 16 axioms. At the top end of the scale, several ontologies contained justifications with very large numbers of axioms. For example Ontologies 48, 50, 63, 18 and 41 contained justifications with 21, 23, 24, 25 and 37 axioms respectively. Finally it should be noted that these larger justifications do not simply consist of long chains of atomic subclass axioms. All in all, the number of justifications per entailment, and the size of justifications points to considerable logical richness being present in many ontologies in the BioPortal corpus.

6.2 Finding Single Justifications

Experiment 1 has shown that it is largely practical to compute all justifications for entailments in real ontologies using model based diagnosis techniques. A key feature of this technique is that while it is used for computing all justifications, it makes use of a pluggable subroutine for computing single justifications. Therefore, the experiments that follow centre around computing single justifications for entailments and a comparison of the high level methods of doing this. Specifically, the effectiveness of the expand stage is examined in the context of a black-box expand-contact algorithm for finding a single justification. In essence Algorithm 4.2 is investigated with three different expansion routines: (1) Algorithm 4.5

(Expansion by Selection Function), (2) Algorithm 4.3 (All in One Expansion, which essentially no expansion phase) and (3) Algorithm 4.4 (Glass-Box Tracing based Expansion). The divide and conquer contraction subroutine was used for all sub-experiments because both Frierich [FS05], Shchekotykhin [SFJ08] and Suntisrivaraporn [Sun09] carried out experiments which clearly show the benefits of using divide and conquer over other strategies such as the sliding window technique [Kal06].

Experiment 2: Black Box Find One with Different Expansion Techniques

Experiment 2 Algorithm Implementation

Algorithms 4.2, 4.6, 4.5, 4.3 and 4.4 were implemented in Java. Three sub-experiments were run by pluging together the above algorithms in different combinations. First, Algorithms 4.2, 4.6, 4.5 (Expansion by Selection Function) were combined and run on the test data. This was followed by the combination of Algorithms 4.2, 4.6, 4.3 (All in One Expansion i.e. No incremental expansion) and finally, the combination of Algorithms 4.2, 4.6, 4.4 (Glass-Box Tracing). Entailment checking was performed using Pellet 2.2.2, with each entailment check consisting of load of the set of given axioms followed by a query to see if the entailment holds.

Experiment 2 Test Data

For each ontology in the 72 BioPortal ontologies a *random* sample of 500 non-trivial entailments (direct atomic concept inclusion axioms and direct atomic concept assertion axioms) was drawn from the complete set of non-trivial entailments.

Experiment 2 Method

For each combination of subroutines described above, Algorithm 4.1 (Find *All*) was used to repeatedly make calls to Algorithm 4.2 so that multiple justifications could be found for each entailment. A maximum of 50 justifications, and hence 50 calls to the find-one subroutine were made for each sampled non-trivial entailment. In each call the CPU time to find a single justification was recorded.

The experiments were performed on MacBook Pro with a 3.06 GHz Intel Core 2 Duo Processor. The Java Virtual Machine was allocated a maximum of 4 GB of RAM.

Experiment 2 Results

Figures 6.4 and 6.5 show percentile and mean times for computing single justifications using black-box selection function (Algorithm 4.5) and glass-box (Algorithm 4.4) expansion respectively. It should be noted that, in both cases, the 99th percentile times are used to order the ontologies along the x-axis of each plot. Times for black-box with all-in-one expansion, i.e. without incremental expansion, are not shown explicitly. Instead, Figure 6.6 shows the percentage differences in mean times between the mean values for selection function based incremental expansion, and mean times for all-in-one non-incremental expansion.

In the cases where Algorithm 4.5 (expansion by selection function) and Algorithm 4.4 (expansion by glass-box tracing) were used as sub-routines, all calls made to Algorithm 4.2 to find a single justification succeeded for all tested entailments in all ontologies except Ontology 32. Recall from Experiment 1 that this ontology was problematic and incurred a 75 percent failure proportion for the set of atomic class inclusion entailments. When Algorithm 4.3 (all-in-one expansion) was used as a subroutine for Algorithm 4.2 several failures occurred for Ontology 49. Out of 1431 runs of Algorithm 4.2 there were 15 failures. However, all of these failures were caused by stack overflow errors in Pellet.

Experiment 2 Analysis

Since Ontology 32 has proved to be somewhat of a pathological test ontology, in that the main problems with Ontology 32 arise due to poor entailment checking performance rather than inherent issues with justification finding algorithms, it is omitted from the analysis that follows when discussing the practicalities of single justification finding.

The Practicabilities of Finding Single Justifications As expected, the algorithms for finding single justifications work well in practice. Reasoner bugs aside, all three subroutine variants succeed in computing single justifications for all runs of Algorithm 4.2 on all entailments in all ontologies. For the vast majority of ontologies it was possible to compute single justification for all entailments

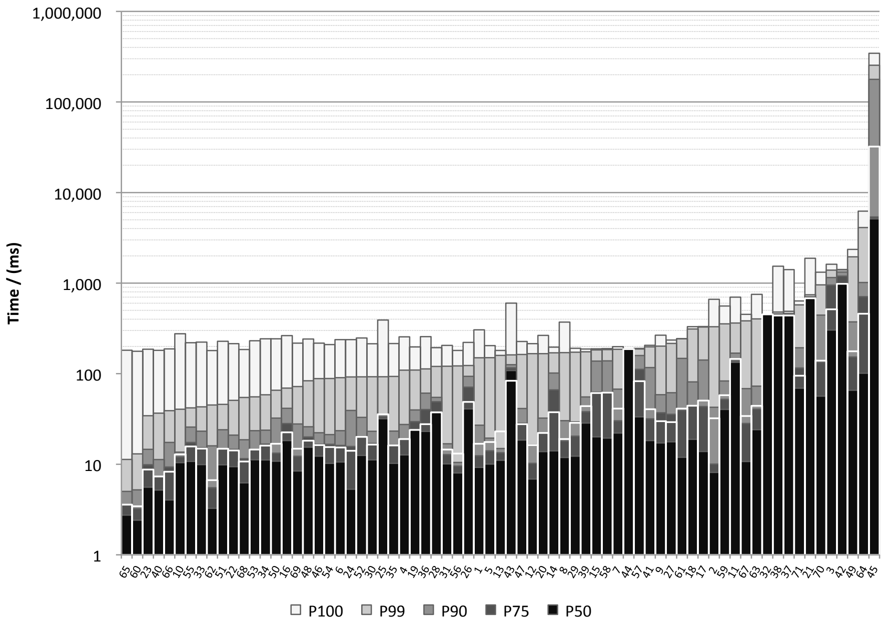


Figure 6.4: The Mean and Percentile Times for Finding One Justification Using Algorithm 4.2 (Expand-Contract) with Algorithm 4.5 (Expansion by Selection Function) and Algorithm 4.6 (Divide and Conquer Contraction) as sub-routines. Mean times are shown in a white outline.

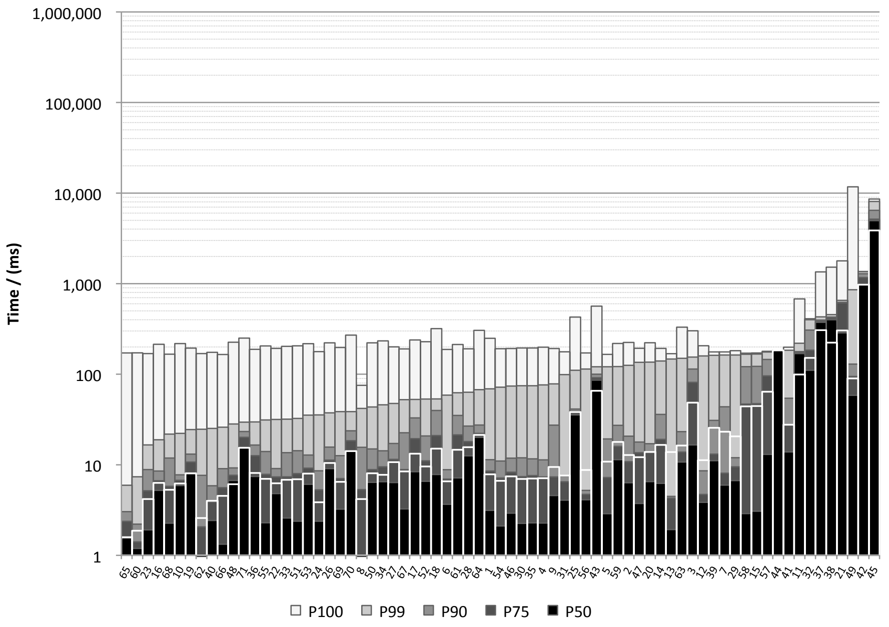


Figure 6.5: The Mean and Percentile Times for Finding One Justification Using Algorithm 4.2 (Expand-Contract) with Algorithm 4.4 (Expansion by Glass Box Tracing) and Algorithm 4.6 (Divide and Conquer Contraction) as sub-routines. Mean times are shown in a white outline.

well within the 1 second boundary. As can be seen by comparing Figure 6.4 with Figure 6.5, over the whole BioPortal corpus, the glass-box expansion based algorithm (Algorithm 4.4) performed *slightly* better than the black-box incremental expansion based algorithm (Algorithm 4.5)—for the majority of ontologies the mean times and 99th percentile times were only slightly less (typically tens of milliseconds) for the glass-box based algorithm than they were for the black-box based algorithm. Indeed, for *most* entailments in *most* ontologies there was not an order of magnitude or more difference in time, with glass-box times on average being 2.89 times less than black-box times (SD = 2.81). There were only three ontologies (Ontology 64, Ontology 3, and Ontology 70) where the mean glass-box times were around an order of magnitude less (21.47 times less, 10.59 times less and 9.84 times less) than mean pure black-box times (mean times for the black-box algorithm being 459.60 ms 514.03 ms and 139.49 ms per justification respectively). Additionally, for the hardest 10 percent of entailments the glass-box times were an order of magnitude less. For the black-box based algorithm there was just one ontology, 45, where 25 percent of entailments took longer than 10 seconds, with the hardest 1 percent of entailments taking just over 120 seconds. The glass-box algorithm fared much better for the hardest cases in this ontology, with all runs taking less than 10 seconds. All in all this is perfectly respectable performance for use in an ontology development environment and for use as a subroutine when computing all justifications. It is fare to say that the performance of optimised black-box single justification finding algorithms is comparable with glass-box algorithms.

The Efficacy of Incremental Expansion The effect of incremental expansion (through the use of a selection function) is depicted in Figure 6.6, which shows the *percentage difference* in mean time due to *not* using the incremental expansion selection function in Algorithm 4.5. As can be seen, there were 12 ontologies in the BioPortal corpus where the expansion function has a negative impact on the performance of Algorithm 4.2. For the remaining 60 ontologies the expansion function has a positive impact on performance to a lesser or greater degree. For the vast majority of ontologies the increase in performance was negligible. However, for 4 of the ontologies, the performance of computing single justifications was boosted between two and three times, for 2 ontologies between

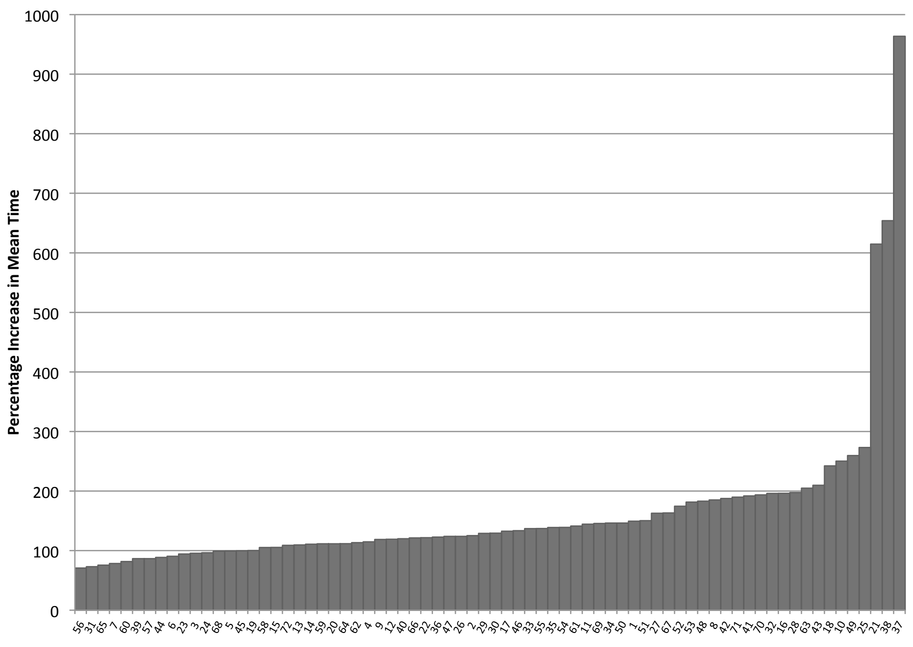


Figure 6.6: Mean Times to Compute Single Justifications Using Algorithm (Expand-Contract Find One with Algorithm 4.3 (All In One Expansion) and Algorithm 4.6 (Divide and Conquer) as sub-routines. Times are expressed as a percentage of the mean times for computing single justifications using Algorithm 4.5 in place of Algorithm 4.3.

6 and 7 times, and for one ontology almost 10 times. While the gains in performance are not of orders of magnitude or more, they are modest improvements. Moreover, since incremental expansion does not appear to have a massively negative effect, and the fact that it is relatively cheap to implement, suggest that it is a worthwhile optimisation.

The Usefulness of Glass-Box Expansion As discussed above, in the experiments here the glass-box expansion subroutine provided some benefits, namely an order of magnitude improvement in time to compute single justifications for 1-10 percent of entailments in three ontologies. Closer inspection revealed that these benefits were achieved because fewer entailment checks were required in both the expansion phase (glass-box only requires one entailment check here) and in the contraction phase, as the axioms returned from the expansion phase usually correspond to the justification or a few axioms over the justification. If available, glass-box expansion is a worthwhile optimisation that should be used. However, the obvious disadvantage of glass-box tracing is that it does not work with reasoners that do not support it. Out of FaCT++, JFaCT, HermiT, Pellet and Racer (the mainstream OWL reasoners), only Pellet supports glass-box tracing. This perhaps due to the fact that the burden of augmenting an existing reasoner with tracing is rather high—certainly substantially higher than the burden of implementing a black-box justification finding algorithm. All things considered, especially given the rather good performance of black-box algorithms, it is questionable as to whether the performance boosts that glass-box tracing provides outweigh the cost of implementation.

6.3 Justifications For Inconsistent Ontologies

In what follows the feasibility of using Algorithm 4.1 to compute justifications for *inconsistent ontologies* is investigated. Since inconsistent ontologies do not have "key" entailments that can provide seed signatures it is not possible to use the incremental expansion by selection function subroutine with them. Hence, Algorithm 4.1 is only used with two different sets of sub-routines, namely Algorithms 4.3 (No Incremental Expansion) and 4.6 (Divide and Conquer), and Algorithms 4.4 (Glass-box Tracing Expansion) and 4.6 (Divide and Conquer). The first combination of subroutines provides a pure black-box based approach,

Table 6.2: Additional Inconsistent Ontologies

Ontology	Name	Expressivity	Axioms
F	assignment	$\mathcal{ALCIQ}$	88
G	boat	$\mathcal{ALCIF}$	22
H	classes-as-values	$\mathcal{ALCO}$	13
I	compara-grid-ontology	$\mathcal{ALCHIQ}$	409
J	country	$\mathcal{SHOIQ}$	5921
K	fish	$\mathcal{ALCH}$	49
L	idbexport	$\mathcal{SHOIN}$	2417
M	microdemo	$\mathcal{SHOIQ}$	1458
N	pizza	$\mathcal{SHOIQ}$	179
O	spectro	$\mathcal{ALCON}$	26612
P	travel	$\mathcal{SHOIQ}$	6437
Q	units	$\mathcal{ALCOIF}$	2254

while the second set of subroutines provides a hybrid glass-box based approach.

Experiment 3: Justifications For Inconsistent Ontologies

The following experiment investigates computing all justifications for the five inconsistent ontologies that are contained in the BioPortal corpus plus twelve inconsistent ontologies that are *not* contained in BioPortal. In general, people do not publish inconsistent ontologies which makes them difficult to obtain for testing purposes. In fact, it is rather surprising that BioPortal contains inconsistent ontologies at all. It is possible that the authors of these ontologies did not use a reasoner for consistency checking, or they used a tool chain which results in lossy communication between editor and reasoner and inconsistencies did not get flagged.[1]

Experiment 3 Algorithm Implementation

The algorithm implementation consisted of Algorithm 4.1 for finding all justifications, which used Algorithm 4.2, Algorithm 4.3 (All-In-One Expansion) and Algorithm 4.6 (Divide and Conquer Contraction) as subroutines. Pellet 2.2.2 was used throughout as, in addition to supporting glass-box tracing, it was found to be the most robust and best performing reasoner for checking consistency in the BioPortal corpus. Each entailment check consisted of a load followed by a query to see if the set of axioms loaded was consistent.

[1]Some early versions of Protégé-OWL (3.x) do not transmit certain types of axioms and concept constructors which are not handled by the DIG [BMC03] language to the reasoner.

Table 6.3: Inconsistent Ontology Entailment Check Times

		Consistency Check Time		
Ont.	**Name**	**Load (ms)**	**Query (ms)**	**Total (ms)**
A	bioportal-metadata	193	479	**672**
B	nifstd	4,332	807	**5,139**
C	ont.-for-disease-genetic-inv.	270	159	**429**
D	suggested-ont.-for-pharm.	1,697	460	**2,157**
E	sysmo-jerm	163	194	**357**
F	assignment	145	942	**1,087**
G	boat	124	32	**156**
H	classes-as-values	122	25	**148**
I	compara-grid-ontology	162	63	**226**
J	country	541	204	**745**
K	fish	127	67	**195**
L	idbexport	314	129	**443**
M	microdemo	270	186	**457**
N	pizza	143	94	**238**
O	spectro	742	189	**932**
P	travel	974	232	**1,206**
Q	unit	271	83	**354**

Experiment 3 Test Data

The BioPortal contained five ontologies that were inconsistent. In addition twelve external ontologies shown in Table 6.2, which were obtained from public mailing lists, were used. As can be seen, these ontologies vary considerably in size and expressivity. The complete set of inconsistent ontologies is listed in Table 6.3 which shows consistency checking times using Pellet 2.2.2 (broken down into load and query times) for each ontology. Note that the ontologies have been labelled with upper case letters to distinguish them from the set of 72 consistent ontologies.

Experiment 3 Method

The implementations of Algorithm 4.1 and each combination of subroutines was run for each ontology listed in Table 6.3. Due to practical considerations a time out of 30 minutes of CPU time was imposed on each run, and as many justifications as possible were computed within this 30 minutes. The CPU time to compute each justification was measured, along with the number of justifications computed within 30 minutes of CPU time, and the CPU time required to find all justifications if all were found. The experiments were performed on MacBook Pro with a 3.06 GHz Intel Core 2 Duo Processor. The Java Virtual Machine was allocated a maximum of 4 GB of RAM.

Table 6.4: Inconsistent Ontology Results

	Found All		**Found**		**Mean Size**		**Time All (ms)**		**Time Single (ms)**	
Ont.	**BB**	**GB**	**BB**	**GB**	**BB**	**GB**	**BB**	**GB**	**BB**	**GB**
A	✔	✔	2	2	12	12	6,066	2,404	378	36
B			7	158	8.71	9.49	-	-	200,229	4,144
C			115	122	17.09	17.16	-	-	909	53
D			10	90	12.6	12.4	-	-	105,917	1,529
E	✔	✔	366	366	9.81	9.81	166,959	65,902	405	8
F			1	1	8	9	-	-	193,024	1,428
G	✔	✔	1	1	5	5	306	226	50	5
H	✔	✔	2	2	2	2	215	199	34	3
I	✔	✔	9	9	13.11	13.11	6,212	1,397	206	15
J	✔	✔	4	4	10.75	10.75	70,588	45,146	564	194
K	✔	✔	183	183	16.81	16.81	24,319	12,383	100	2
L	✔	✔	2	2	4.5	4.5	4,796	2,136	684	173
M	✔	✔	1	1	11	11	19,644	5,793	1,636	57
N	✔	✔	3	3	5.33	5.33	4,646	1,697	386	48
O			161	162	2.02	2.02	-	-	725	136
P	✔	✔	4	4	3	3	12,664	9,868	602	228
Q			118	118	2	2	-	-	215	40

Experiment 3 Results

Results are shown in Table 6.4, where GB stands for Glass-Box and BB stands for Black-Box. Columns 2 and 3 use a tick to indicate that all justifications for the inconsistency were found. Columns 4–7 show the number and mean size of justifications computed in 30 minutes of CPU time. The right most four columns show times to compute all justifications (if all were found) and the mean time to compute a single justification.

Experiment 3 Analysis

The Practicality of Computing All Justifications As can be seen from Table 6.4, there were 11 out of the 17 ontologies where all justifications for the inconsistency could be computed within 30 minutes of CPU time and 6 ontologies where not all justifications could be computed within this time limit—this result was the same for both black-box and glass-box algorithms. In most of these cases all justifications were computed well within the 30 minute time boundary. The failures are discussed below, however, for the successful ontologies the maximum time occurred for Ontology E, where there were 366 justifications for the inconsistency, which required a time of just over 166 seconds for the black-box implementation and 45 seconds for the glass-box implementation. This was followed by Ontology-J, with 183 justifications and 70.6 seconds and 45.1 seconds for black-box and glass-box respectively. All other ontologies for which all justifications could be computed had times that fell below 25 seconds. These results seem perfectly acceptable for use in ontology development environments and auxiliary services.

Over all, it is noticeable that the glass-box implementation performs better than the black-box implementation (although the difference is not so huge that it is orders of magnitude out). This is to be expected as the glass-box algorithm has the advantage of reducing the search space for the contraction part of the find-all algorithm, which means that on the whole the glass-box based algorithm requires fewer time consuming entailment checks than is required by the black-box algorithm. The difference in time between the glass-box and black-box algorithms is not orders of magnitude, because the benefits of the glass-box algorithm only come into play when actually computing *single* justifications, and most of the time is spent constructing the hitting set trees with plenty of justification reuse taking place during construction.

Reasons for Failure to Compute All Justifications There were 6 ontologies out of the corpus of 17 (3 from BioPortal and 3 external) where not all justifications could be computed within 30 minutes of CPU time using either black-box or glass-box algorithms. These ontologies can be divided into three categories: Category-I—Ontologies where large numbers of justifications were computed by both the black-box and glass-box implementations. Ontologies C, O and Q fall into this category; Category-II—Ontologies where large numbers of justifications were computed by the glass-box implementation, but comparatively few by the black-box implementation. Ontologies B and D fall into this category; and Category-III—Ontologies where few justifications could be computed by both the glass-box and black-box implementations. Ontology F falls into this category.

Category-I is indicative of failure due to large numbers of justifications and the sheer size of the hitting set tree. Indeed, for ontologies in this category the hitting set tree sizes were as follows: Ontology-C = 1.3 million nodes, Ontology-O = 2.35 million nodes, Ontology-Q = 2.38 million nodes. The fact that black-box and glass-box implementations could compute similar or exactly numbers of justifications tends to show that entailment checking performance is not the dominant problem. Indeed, this is backed up by the consistency checking times in Table 6.3.

Category-II is indicative of failure of the glass-box algorithm due to large numbers of justifications, and failure of the black-box algorithm due to entailment checking performance issues. This is because the black-box algorithm typically requires many more checks than the glass-box algorithm and if entailment checking performance is in the order of thousands of milliseconds rather than tens or hundreds of milliseconds the differences can add up. Again, this result is backed up by the entailment checking times in Table 6.3 which shows entailment checking times for ontologies B and D which are an order of magnitude more than other ontologies (5,139 ms and 2,157 ms respectively).

Finally, Category-III is indicative of serious entailment checking performance problems. There was only one ontology in this category, Ontology-F, but both the black-box algorithm and the glass-box algorithm only found 1 justification and timed out within 30 minutes of CPU time. Moreover, the time to find this single justification was extremely large in the black-box case, at 193 seconds, and much smaller in the glass-box case at 1.4 second. This difference is due to entailment checking performance and the fact that glass-box justification finding requires fewer entailment checks than black-box justification finding. Lastly, this ontology exhibited an interesting phenomenon of entailment checking performance being much poorer in subsets of the ontology when compared to the whole ontology. The consistency check for the whole ontology took approximately 1 second, but on average, entailment checking for the black-box algorithm took approximately 2 seconds. In cases like this, glass-box justification finding helps enormously, hence the two order of magnitude difference between black-box and glass-box performance overall.

The Practicality of Computing a Single Justification As can be seen from Table 6.4, it was possible, and practical, to compute *at least one* justification for every inconsistent ontology, for both the black-box and glass-box algorithm and in cases where not all justifications could be computed. For most ontologies, the time required to find one justification was around 1 second or less than a second for both glass-box and black-box implementations. This is obviously acceptable performance for on demand debugging in ontology development environments. Three ontologies stand out for taking hundreds of seconds instead of hundreds of milliseconds when operated on by the black-box implementation. These are Ontology B, Ontology D and Ontology F. As described above, the evidence points to problems with entailment checking performance, and as can be seen from Table 6.3, all three ontologies have consistency checking times that are much longer than the other ontologies. Ontology F is a somewhat pathological case, as subsets of it are harder to check than the whole ontology. Overall though, the good news is that it is possible to compute a single justifications for all ontologies using both the black-box and glass-box algorithms. Being able to compute just one justification for an inconsistent ontology can be of enormous benefit to end users as discussed below.

The Differences Between Black-Box and Glass-Box Algorithms For computing single justifications, it is particularly clear that there is a marked difference between the runtime performance of the black-box algorithm compared to the glass-box algorithm. In most cases there was an order of magnitude difference between the algorithms in CPU time required to compute single justifications. The difference was most pronounced for ontologies B, D, F and M, where the black-box time was three orders of magnitude greater than the glass-box time. This was due to entailment checking performance and the number of entailment checks required by the glass-box implementation, which required fewer compared to the black-box implementation. Obviously, it is preferable that a user wait 4 seconds for a justification rather than 200 seconds, but 200 seconds is still not such a staggeringly long time—particularly when justifications can be so crucial to the task of understanding why an ontology is inconsistent.

In terms of computing all justifications, the differences in CPU time between black-box and glass-box performance are less noticeable that those for single justifications, even for ontologies where the differences in time for single justifications were very high. The main reason for this is that compared to the number of nodes in a hitting set tree there are comparatively few calls to the ComputeSingleJustification subroutine. For example, in Ontology E there are 113,779 nodes in the black-box hitting set tree but only 390 calls to the ComputeSingleJustification subroutine, and out of these only 366 calls resulted in justifications being computed and thus involved multiple entailment checks.

The differences that are striking between the black-box and glass-box implementations are the number of justifications that can be computed in the event that not all justifications can be computed. In particular Ontology B and Ontology D exhibited large differences in 7 verses 158, and 10 versus 90 justifications respectively. Obviously, whether this makes a practical difference in being able to repair the ontology, or being able to enact a high quality repair (from a modelling point of view) is another matter.

The usefulness of justifications for inconsistent ontologies Despite the fact that, in *some cases*, it is not feasible to compute *all* justifications for real inconsistent ontologies, the ability to obtain just one or two justifications can lend a vital insight into why an ontology is inconsistent, and can allow people to commence manual incremental/interactive repairs. Some ontologies are inconsistent due to highly non-obvious reasons, and it is arguable that without automated explanation support, ontology authors would face a hopeless task of trying to figure out the reasons for an ontology being inconsistent in order to arrive at a manual repair. An example of a justification from the Country ontology is shown in Figure 6.7. The ontology authors, who made a request for help in trying to track down the reasons for the inconsistency, indicated that it was highly unlikely that they would have discovered the reason, without considerable difficulty, using manual debugging techniques.

Using justifications for inconsistent ontologies for repair Even though it may not be possible in practice to compute all justifications for a given ontology, with the availability of at least one justification, it is still possible to carry out a manual interactive repair. For example, on closer inspection of the justifications for the travel ontology it was observed that many of them had the form $\{R(a, l), \top \sqsubseteq \forall R.C\}$, where l is a data value not in the value space of C. In fact, structural inspection revealed that there were over 550 of these kinds of justifications for the travel ontology and over 12000 for the Spectro ontology. In order to test the hypothesis that even seeing a handful of justifications is helpful for interactive debugging and can support repair, the travel ontology was modified to produce a new version where all literals in property assertions were typed with the appropriate data type. The new version of the ontology was still inconsistent, but had just six justifications for it being inconsistent, which were computed in 12 seconds and 9 seconds by the black-box and glass-box implementations respectively. Thus, even though it may not be possible to compute all justifications in one go, being able to compute some justifications can provide enough insight in order to understand the reasons for an ontology being inconsistent so that it is possible to perform a manual repair.

Figure 6.7: An small example justification from the large country ontology. The individual ireland is entailed to be a PhysicalEntity (by axioms (1)(6)) and is also entailed to be a PoliticalEntity (by axioms (7)-(11)), PhysicalEntity and PoliticalEntity are disjoint with each other (axiom (12))

$$\mathsf{Island(ireland)} \tag{1}$$
$$\mathsf{Island} \sqsubseteq \mathsf{LandMass} \tag{2}$$
$$\mathsf{LandMass} \sqsubseteq \mathsf{GeographicalFeature} \sqcap \exists \mathsf{hasCoastline.Coastline} \tag{3}$$
$$\mathsf{GeographicalFeature} \sqsubseteq \mathsf{NaturalPhysicalThing} \tag{4}$$
$$\mathsf{NaturalPhysicalThing} \sqsubseteq \mathsf{NaturalEntity} \tag{5}$$
$$\mathsf{NaturalEntity} \sqsubseteq \mathsf{PhysicalEntity} \tag{6}$$
$$\mathsf{landBoundaryOf(unitedKingdomIrelandBorder, ireland)} \tag{7}$$
$$\mathsf{landBoundaryOf} \sqsubseteq \mathsf{hasLandBoundary}^{-} \tag{8}$$
$$\exists \mathsf{hasLandBoundary}.\top \sqsubseteq \mathsf{Country} \tag{9}$$
$$\mathsf{Country} \sqsubseteq \mathsf{AdministrativeDivision} \tag{10}$$
$$\mathsf{AdministrativeDivision} \sqsubseteq \mathsf{PoliticalEntity} \sqcap \exists \mathsf{directPartOf.PoliticalDiv} \tag{11}$$
$$\mathsf{PoliticalEntity} \sqsubseteq \neg \mathsf{PhysicalEntity} \tag{12}$$

The Inconsistent Ontology Justification Landscape On the whole there were large numbers of justifications per inconsistent ontology. Out of the 17 ontologies, there were just three for which there was one justification for the each ontology being inconsistent—one with 8 axioms, one with 5 axioms and one with 11 axioms. Out of the other ontologies which had many justifications and for which all justifications could be computed, one had 183 justifications with an average size of 16.81 axioms, and another 366 justifications with an average size of 9.81 axioms. Overall, the average sizes were what could be considered to be large, for example, there were ontologies with average sizes of 10.75 axioms, 13.11 axioms, 18.17 and 17.09 axioms. Only four ontologies had mean sizes less than 5 axioms and none of these were ontologies from the BioPortal. The large numbers of justifications per ontology and large sizes of justifications indicates some degree of complexity and non-trivialness in the reasons for these ontologies being inconsistent. It is therefore likely that without justifications it would be very difficult to track down and understand the reasons for these ontologies being inconsistent.

6.4 Discussion

The detailed empirical investigation that has been carried out and presented in this chapter provides strong evidence to conclude that computing all justifications for entailments ontologies in the BioPortal corpus of consistent ontologies is practical. That is, in the vast majority of entailments in the majority of ontologies all justifications can be computed in under of 10 minutes of CPU time. This result applies directly to black-box justification finding, but based on the results obtained for glass-box single justification finding it can be extrapolated to glass-box tracing based approaches. In essence, the justification finding algorithms used in the empirical evaluation show good and robust runtime performance on realistic inputs.

In terms of black-box versus glass-box justification finding, for consistent ontologies, the results show that although glass-box justification finding does boost runtime performance, it is not typically boosted by orders of magnitude. Indeed, in the presence of other optimisations, in particular modularisation, it appears that glass-box tracing is only of incremental benefit in the vast majority of cases. When the effort required to implement glass-box tracing, which is reported as being highly non-trivial, is borne in mind, it is questionable as to whether it is a worthwhile optimisation. In the future, more gain might be made by concentrating on boosting entailment performance, particularly in situations where a large number of axioms are loaded into a reasoner but only a handful of them support the entailment in question.

For inconsistent ontologies the picture seems to be slightly more pessimistic. Although the test corpus used in the empirical investigation was smaller than that for consistent ontologies, the evidence appears to show that the number of justifications coupled with lack of optimisations such as modularisation and incremental expansion makes it difficult to compute all numbers of justifications in a good proportion of ontologies. This result applies to both the BioPortal portion of the corpus and to externally gathered inconsistent ontologies. Nevertheless, the provision of some justifications can be enormously beneficial to end users trying to debug inconsistent ontologies, and in the drive to compute all justifications, this aspect should not be forgotten.

When it comes to glass-box tracing for inconsistent ontologies, the difference in performance between glass-box and black-box algorithms is dramatic. In terms of finding single justifications there was at least an order of magnitude improvement in CPU time required for most ontologies. The benefits of glass-box tracing as an optimisation for computing single justifications in inconsistent ontologies, particularly when faced with a lack of other optimisations such as modularity, are clear to see. While the difference in time between black-box and glass-box was not so pronounced when it came to computing all justifications, glass-box tracing appears to be a worthwhile optimisation. The good performance of glass-box tracing suggests that improvements could be made to the black-box algorithm. One lesson that might be learnt from glass-box tracing is the lesson of how an optimised reasoner goes about detecting that an ontology is inconsistent. This information could then be fed into a carefully designed goal directed expansion function for inconsistent ontologies.

For consistent and inconsistent ontologies there were ontologies that contained entailments which had huge numbers of justifications, and justifications which were large in size—upwards of 100, towards 1000 justifications per entailment were found and justifications of 10, 20, 30 axioms in size were found. Ontologies with huge numbers of justifications per entailment like this have not been documented before and the data obtained as part of the experiments detailed here offer a new and interesting picture on the justification landscape for real world ontologies.

Finally, one of the main outstanding issues is whether or not something can be done in situations where at the moment it is not possible to compute all justifications. The evidence shows that this situation will be encountered for some entailments in real ontologies—more in the case of inconsistent ontologies than consistent ontologies. Although entailment checking performance has some part to play in a small number of cases, the main issue lies in the makeup of ontologies and their justificatory structure. Lots of justifications with little overlap can cause problems for the model based diagnosis techniques used in this thesis. In this situation it is difficult to optimise out these problems as they start to butt up against the intractability of the underlying algorithms. For a given entailment, the only way forward would be to somehow break the hitting set tree down into smaller portions, compute smaller numbers of justifications in each portion and then combine these justifications to give the final result. However, it is difficult to know how this can be done. One possibility would be to use some kind of lemmatisation, where lemmas allow smaller sets of justifications to be computed and combined together. However, the ability to generate these lemmas and ensure completeness is obviously highly non-trivial.

6.5 Conclusions

With regards to justifications for entailments in realistic *consistent* ontologies, the large amount of empirical evidence gathered strongly suggests that:

- Finding all justifications for entailments is practical.
- Optimised and robustly implemented algorithms exhibit good performance which is more than likely to be acceptable for use in ontology debugging environments and applications which use justification finding as an auxiliary service.
- For finding all justifications, the performance of optimised black-box algorithms is comparable to the performance of hybrid glass-box algorithms which use tableau tracing.
- While model based diagnosis techniques for computing all justifications fare extremely well, they do have a high worst case complexity, and there are examples of entailments in realistic consistent ontologies for which it is not possible to compute all justifications. In these cases an incomplete solution, which could still consist of hundreds of justifications, must be accepted.
- Black-box and glass-box algorithms for single justification finding are extremely robust. Assuming that entailment checking can be performed by a backing reasoner, it is always possible to compute a single justification for an entailment using both black-box and glass-box techniques on realistic inputs.
- Glass-box algorithms can offer worthwhile optimisations when finding single justifications, but for most realistic ontologies the performance gains compared to optimised black-box reasoners appear to by less than an order of magnitude of time.
- The number of justifications for entailments in realistic ontologies can be very high, peaking at around 1000 justifications per entailment. The sizes of justifications in these ontologies can be very large, peaking at around 40 axioms per justification. The majority of ontologies with non-trivial entailments have multiple justifications per entailment with multiple axioms per justification.

In terms of justifications for realistic inconsistent ontologies, the empirical evidence strongly suggests that:

- It is always possible to compute a single justification for an inconsistent ontology using both black-box and glass-box algorithms.
- For single justification finding, glass-box tracing can offer huge performance increases of up to three orders of magnitude over pure black-box algorithms when working with inconsistent ontologies.
- It is not always possible to compute all justifications for an inconsistent ontology and this problem seems to be more prevalent than it does for entailments in consistent ontologies. In these cases, the number and size of justifications strongly suggest that it necessary to accept some degree of incompleteness.
- Justifications are typically numerous, can be large in size and are non-trivial.

Chapter 7

Justification Granularity

A key aspect of justifications is that they operate on the level of *asserted* axioms. That is, the axioms in a justification directly correspond to axioms contained in some ontology. When people write axioms in tools such as Protégé-4 they typically write axioms that contain multiple nested class expressions. In part, the rich array of class constructors provided by OWL permits and encourages this, but even lightweight ontology languages, such as $\mathcal{EL}$, or languages that merely contain conjunction and disjunction such as $\mathcal{HL}$, allow class expressions to be nested to arbitrary depths. The upshot of this, is that justifications can be rather coarse-grained explanation devices. In essence, justifications do not identify *parts* of axioms responsible for an entailment. This has ramifications for both entailment comprehension and repair. The main problems that are encountered when working with justifications, where axioms are the basic unity of currency, are outlined below.

Axioms in a Justification can contain superfluous parts While all axioms in a justification are required to support the entailment in question, it may be the case that not all *parts of axioms* are required to support of the entailment. That is, axioms may contain *superfluous* parts. Consider the justification shown in Figure 7.1, which is a justification taken from the myGrid [WAH+07] ontology for the entailment $\mathsf{DataIdentifier} \sqsubseteq \mathsf{Identifier}$. This particular justification contains several axioms with parts that are superfluous for the entailment, which are shown in Figure 7.1 using strikethrough. As can be seen, the nested existential restriction, $\exists$ $\mathsf{isIdentityOf.Data}$, in the first axiom is superfluous for the entailment. Similarly, the existential restrictions in the second and fourth

$$\begin{aligned}\mathcal{J} = \{&\mathsf{DataIdentifier} \equiv \mathsf{Metadata} \sqcap (\exists\ \mathsf{encodes}.(\mathsf{Identity}\ \cancel{\sqcap (\exists\ \mathsf{isIdentityOf.Data})}))\\ &\mathsf{Metadata} \equiv \mathsf{Data}\ \cancel{\sqcap (\exists\ \mathsf{encodes.InformaticsModifierConcept}})\\ &\mathsf{Data} \sqsubseteq \mathsf{InformaticsPhysicalStructure}\\ &\mathsf{InformaticsPhysicalStructure} \equiv \mathsf{PhysicalStructure}\ \cancel{\sqcap (\exists\ \mathsf{inDomain.InformaticsDomain})}\\ &\mathsf{Identifier} \equiv \mathsf{PhysicalStructure} \sqcap (\exists\ \mathsf{encodes.Identity})\}\\ &\qquad\qquad\qquad \models \mathsf{DataIdentifier} \sqsubseteq \mathsf{Identifier}\end{aligned}$$

Figure 7.1: An Example of Superfluity

axioms are also superfluous conjuncts as far as the entailment is concerned. In addition to parts of concepts, there is another level of superfluity in the axioms in $\mathcal{J}$, namely the fact that several axioms are equivalent concept axioms (of the form $C \equiv D$) but, in each case, only one direction of implication is needed (i.e. $C \sqsubseteq D$ or $D \sqsubseteq C$). For example, given the last axiom, it is simply sufficient to know that $\mathsf{PhysicalStructure} \sqcap (\exists\ \mathsf{encodes.Identity}) \sqsubseteq \mathsf{Identifier}$ rather than $\mathsf{Identifier} \sqsubseteq \mathsf{PhysicalStructure} \sqcap (\exists\ \mathsf{encodes.Identity})$. Superfluous parts are problematic because they can *distract* a person from understanding a justification. Without any highlighting, such as strikeout, it can be hard to separate out the parts which are superfluous from the parts which matter for the entailment. With equivalent concept axioms it can be difficult to know which way round to read the axiom if only one direction is required but both directions are included. It is easy to imagine that focusing a persons's attention on the relevant parts of an axioms in a justification can make it quicker and easier for them to understand that justification.

A justification can mask multiple reasons for an entailment Within a single justification there can be multiple ways in which the parts of axioms combine together to produce the entailment in question. For example, consider the justification shown in Figure 7.2, which is taken from the Sequence Ontology (SO) [ELM$^+$05, MBE11]. This is a justification for the entailment $\mathsf{EngForeignGene} \sqsubseteq \mathsf{EngineeredForeignRegion}$. Within this one justification there are two *distinct reasons* as to why this entailment holds. Both reasons revolve around $\mathsf{EngineeredForeignRegion}$ being a subclass of $\exists\ \mathsf{hasQuality.Engineered}$. As indicated by underlining in Figure 7.2, the intermediate entailment $\mathsf{EngForeignGene} \sqsubseteq \exists\ \mathsf{hasQuality.Engineered}$ can be derived from parts of the first axiom alone, or

$$\begin{aligned}\mathcal{J} = \{&\underline{\mathsf{EngForeignGene}} \equiv \mathsf{EngineeredGene} \sqcap \exists\ \mathsf{hasQuality.Foreign} \sqcap \underline{\exists\ \mathsf{hasQuality.Engineered}}\\ &\mathsf{EngineeredGene} \equiv \mathsf{Gene} \sqcap \underline{\exists\ \mathsf{hasQuality.Engineered}}\\ &\mathsf{Gene} \sqsubseteq \mathsf{BiologicalRegion}\\ &\mathsf{BiologicalRegion} \sqsubseteq \mathsf{Region}\\ &\mathsf{EngineeredForeignRegion} \equiv \mathsf{Region} \sqcap \exists\ \mathsf{hasQuality.Engineered} \sqcap \exists\ \mathsf{hasQuality.Foreign}\}\\ &\qquad\qquad \models \mathsf{EngForeignGene} \sqsubseteq \mathsf{EngineeredForeignRegion}\end{aligned}$$

Figure 7.2: An Example of Internal Masking

parts of the first and second axioms. This is due to the fact that first axiom entails $\mathsf{EngForeignGene} \sqsubseteq \mathsf{EngineeredGene}$ by itself, while the second axiom entails $\mathsf{EngineeredGene} \sqsubseteq \exists\mathsf{hasQuality.Engineered}$. This thesis defines the phenomena of a single justification containing multiple reasons for an entailment as *internal masking*. The term "reason" will be made more precise later in the thesis, but roughly speaking, a reason is a minimal way in which *parts* of axioms may combine to produce an entailment. The presence of multiple reasons within a single justification is indicative of superfluity, and as described above, such superfluity can cause problems with understanding.

A justification can mask relevant axioms Consider the ontology $\mathcal{O}$ shown in Figure 7.3, which is a subset of the Experimental Factor Ontology (EFO) [MRBP08, MHA+10]. This ontology entails $\mathsf{Emphysema} \sqsubseteq \mathsf{RespiratorySystemDisease}$ and contains exactly *one* justification for this entailment, namely $\mathcal{J}$ in Figure 7.3. On closer inspection, the second axiom in $\mathcal{O}$ also plays some role in this entailment. However, there is no justification for $\mathcal{O} \models \mathsf{Emphysema} \sqsubseteq \mathsf{RespiratorySystemDisease}$ that includes the second axiom: $\mathsf{Emphysema} \sqsubseteq \forall\ \mathsf{hasDiseaseLocation}\ .\ \mathsf{Lung}$. If the filler Lung is discarded from the first axiom to weaken it to $\mathsf{Emphysema} \sqsubseteq \exists\mathsf{hasDiseaseLocation}.\top$ then this combines with the second axiom in $\mathcal{O}$ to result in $\mathcal{J}'$ in Figure 7.3, which *does* contain the "masked" axiom $\mathsf{Emphysema} \sqsubseteq \forall\ \mathsf{hasDiseaseLocation.Lung}$. It is likely that people working with this ontology and looking at justifications for $\mathcal{O} \models \mathsf{Emphysema} \sqsubseteq \mathsf{RespiratorySystemDisease}$ would be completely unaware of this second "external" reason. This thesis defines this phenomena as *external masking*.

$$\begin{aligned}
\mathcal{O} = \{&\mathsf{Emphysema} \sqsubseteq \exists\ \mathsf{hasDiseaseLocation.Lung}\\
&\underline{\mathsf{Emphysema} \sqsubseteq \forall\ \mathsf{hasDiseaseLocation.Lung}}\\
&\mathsf{Lung} \sqsubseteq \mathsf{RespiratorySystemStructure}\\
&\mathsf{RespiratorySystemDisease} \equiv \exists\ \mathsf{hasDiseaseLocation.RespiratorySystemStructure}\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{J} = \{&\mathsf{Emphysema} \sqsubseteq \exists\ \mathsf{hasDiseaseLocation.Lung}\\
&\mathsf{Lung} \sqsubseteq \mathsf{RespiratorySystemStructure}\\
&\mathsf{RespiratorySystemDisease} \equiv \exists\ \mathsf{hasDiseaseLocation.RespiratorySystemStructure}\}\\
&\qquad \models \mathsf{Emphysema} \sqsubseteq \mathsf{RespiratorySystemDisease}
\end{aligned}$$

$$\begin{aligned}
\mathcal{J}' = \{&\mathsf{Emphysema} \sqsubseteq \exists\ \mathsf{hasDiseaseLocation}.\underline{\top}\\
&\underline{\mathsf{Emphysema} \sqsubseteq \forall\ \mathsf{hasDiseaseLocation.Lung}}\\
&\mathsf{Lung} \sqsubseteq \mathsf{RespiratorySystemStructure}\\
&\mathsf{RespiratorySystemDisease} \equiv \exists\ \mathsf{hasDiseaseLocation.RespiratorySystemStructure}\}\\
&\qquad \models \mathsf{Emphysema} \sqsubseteq \mathsf{RespiratorySystemDisease}
\end{aligned}$$

Figure 7.3: An Example of External Masking

Multiple justifications can mask a shared core When considering parts of axioms two or more justifications may share a common "core". Figure 7.4 shows an example of this phenomena. In this example, which is taken from the International Classification of Nursing Practice (ICNP) ontology [BCH06], there are two justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ for the entailment $\mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{FamilyAttitude}$. As indicated by the underlining in Figure 7.4, when the superfluous parts of axioms in $\mathcal{J}_1$ and $\mathcal{J}_2$ are ignored, $\mathcal{J}_2$ represents the same reason for the entailment as $\mathcal{J}_1$. In other words they share the same "core", and are only distinguishable because different axioms are "decorated" with superfluous parts. For example, the second axiom in $\mathcal{J}_1$ contains the conjunct $\exists\ \mathsf{hasPotentialityState.Actual}$, but the corresponding axiom in $\mathcal{J}_2$ does not contain any superfluous parts. It is easy to see that highlighting shared cores across multiple justifications would make it much easier and quicker for a person to work through these justifications. This thesis defines this phenomena as *shared core masking.*

Justifications can result in over-repair Finally, a theme running through all of the above examples, is the issue of *repair*. As described in Section 2.3.3 on

$$
\begin{aligned}
\mathcal{J}_1 = \{ & \underline{\mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{ActualNegativeFamilyProcess}} \\
& \underline{\mathsf{ActualNegativeFamilyProcess} \equiv \mathsf{NegativeFamilyProcess}} \sqcap (\exists\ \mathsf{hasPotentialityState.Actual}) \\
& \underline{\mathsf{NegativeFamilyProcess} \sqsubseteq \mathsf{FamilyProcess}} \\
& \underline{\mathsf{FamilyProcess} \equiv \mathsf{Process} \sqcap (\exists\ \mathsf{isPerformedBy.Family})} \\
& \underline{\mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{NegativeAttitude}} \\
& \underline{\mathsf{NegativeAttitude} \sqsubseteq \mathsf{Attitude}} \\
& \underline{\mathsf{FamilyAttitude} \equiv \mathsf{Attitude} \sqcap (\exists\ \mathsf{isPerformedBy.Family}}\} \\
& \qquad \models \mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{FamilyAttitude}
\end{aligned}
$$

$$
\begin{aligned}
J_2 = \{ & \underline{\mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{ActualNegativeFamilyProcess}} \\
& \underline{\mathsf{ActualNegativeFamilyProcess} \sqsubseteq \mathsf{NegativeFamilyProcess}} \\
& \underline{\mathsf{NegativeFamilyProcess} \equiv \mathsf{FamilyProcess}} \sqcap (\exists\ \mathsf{hasAbsoluteJudgedState.NegativeJudgedState}) \\
& \underline{\mathsf{FamilyProcess} \equiv \mathsf{Process} \sqcap (\exists\ \mathsf{isPerformedBy.Family})} \\
& \underline{\mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{NegativeAttitude}} \\
& \underline{\mathsf{NegativeAttitude} \equiv \mathsf{Attitude}} \sqcap (\exists\ \mathsf{hasAbsoluteJudgedState.NegativeJudgedState}) \\
& \underline{\mathsf{FamilyAttitude} \equiv \mathsf{Attitude} \sqcap (\exists\ \mathsf{isPerformedBy.Family})}\} \\
& \qquad \models \mathsf{ActualNegativeFamilyAttitude} \sqsubseteq \mathsf{FamilyAttitude}
\end{aligned}
$$

Figure 7.4: An Example of Shared Cores

Page 47, a simple repair that breaks some entailment is based on choosing exactly one axiom from each justification for that entailment and removing these chosen axioms from the associated ontology. With this style of repair, there are some basic notions of minimality. Given a set of justifications $\mathcal{J}_1, \ldots, \mathcal{J}_n$ for $\mathcal{O} \models \eta$, a repair $\mathcal{R} \subseteq \mathcal{O}$ (Definition 3, Page 47) can be minimal in the following senses:

- **Justification Minimal** For each $\mathcal{J}_i$, $|\mathcal{R} \cap \mathcal{J}_i = 1|$. In other words, $\mathcal{R}$ contains one and only one axiom per justification. If $\mathcal{R}$ is justification minimal then upon repair there are no axioms that are removed from $\mathcal{O}$ that do not directly contribute to the repair.

- **Cardinality Minimal** A repair is cardinality minimal (Definition 4, Page 47), if there is no $\mathcal{R}' \subseteq \mathcal{O}$ such that $|\mathcal{R}'| < |\mathcal{R}|$. If $\mathcal{R}$ is cardinality minimal then the fewest number of axioms is removed from $\mathcal{O}$ upon repair. In the model diagnosis arena this corresponds to a diagnosis.

- **Semantically Minimal** Any given repair essentially removes some entailments from an ontology, a semantically minimal repair removes the fewest entailments from $\mathcal{O}$, for some class of entailments such as atomic subsumptions.

However, these notions of minimality are rather basic, and it should be clear that, when working with "regular" justifications and whole axioms, there is a potential to "over repair" an ontology so that more entailments are lost than is necessary. For example, removing the first axiom from $\mathcal{J}$ in Figure 7.2, would break the entailment $\mathsf{EngForeignGene} \sqsubseteq \mathsf{EngineeredForeignRegion}$, and it might be minimal in the first two senses of minimality above, but it would also cause the entailment $\mathsf{EngForeignGene} \sqsubseteq \mathsf{hasQuality.Foreign}$ to be lost, which might be undesirable. Further more, with the effects of masking, not all reasons for an entailment may be obvious, so if a desired repair involved "hacking" and rewriting axioms, the repair may not actually be successful. For example, returning to Figure 7.2, if the first axiom in $\mathcal{J}$ was rewritten to $\mathsf{EngForeignGene} \sqsubseteq \mathsf{EngineeredGene} \sqcap \exists\ \mathsf{hasQuality.Foreign}$ then the original entailment would still hold. In essence, the effects of superfluity and masking can cause over-repair or result in a failed repair. It is therefore desirable that any definition of fine-grained justifications should result in justifications that make it easy to devise and enact *fine-grained repairs.*

Summary In summary, this chapter addresses the above issues with fine-grained justifications. The purpose of this chapter is to identify the desirable characteristics of fine-grained justifications, propose proper definitions for these types of justifications, and show how they can be computed for entailments in real ontologies.

7.1 Related Work

In what follows, approaches to defining and computing fine-grained justifications are discussed. Work that directly relates to explanation of entailments in Description Logics and OWL is first presented, and then more general, but related techniques that are of interest are mentioned.

7.1.1 Prior Approaches

Axiom Splitting and Flattening In [KPG06], Kalyanpur et al. propose definitions and present algorithms for computing "precise" justifications. Given an ontology $\mathcal{O} \models \eta$, they define a precise justification to be a justification for η with respect to $\sigma(\mathcal{O})$, where σ is a *splitting* function. Kalyanpur describes the goal of the splitting function as to split a "TBox $\mathcal{T}$ into 'smaller' axioms to obtain an equivalent TBox $\mathcal{T}_s$ that contains as many axioms as possible.". The splitting function is defined both for $\mathcal{SHOIQ}$ concepts, which are assumed to be in negation normal form, and for $\mathcal{SHOIQ}$ axioms. A key part of the splitting process is the introduction of fresh concept names, which are used in order to split "axioms into smaller sizes". For example (taken from [KPG06]), the axiom $A \sqsubseteq \exists R.(C \sqcup D)$, is split into $A \sqsubseteq \exists R.X$, $X \sqsubseteq C \sqcup D$, $C \sqsubseteq X$ and $D \sqsubseteq X$. The last three axioms come from the splitting of the definition $X \equiv C \sqcup D$, which gets introduced to flatten the nested disjunct in the filler of existential restriction in the original axiom.

Another interesting aspect of the splitting function is the way that it generates the cross-product of disjunctions. For example applying the splitting function to $(A \sqcap B) \sqcup (C \sqcap D)$ results in $\{(A \sqcup C), (A \sqcup D), (B \sqcup C), (B \sqcup D)\}$. Also, the fillers of universal and existential restrictions get special treatment via a "Π operator", which "applies a set of concepts to a preceding restriction operator", for example splitting $\forall R.(B \sqcap (C \sqcup D))$ gives $\{\forall R.B, \forall R.(C \sqcup D)$.

Curiously, for a TBox $\mathcal{T}$, the definition of the splitting function normalises all GCIs in $\mathcal{T}$ of the form $C \sqsubseteq D$ into $\top \sqsubseteq \neg C \sqcup D$. It then applies the splitting function for class expressions on the right hand side of these axioms to produce $\mathcal{T}_S$. However, all of the examples given in the paper ignore this normalisation step, for example, $A \sqsubseteq B \sqcap C$ is split into $A \sqsubseteq B$ and $A \sqsubseteq C$ rather than a splitting of $\top \sqsubseteq \neg A \sqcup (B \sqcap C)$.

Although not part of the formal definitions in [KPG06], Kalyanpur sketches out some post-processing aspects and issues relating to usability. In particular, given a set of precise justifications for an entailment, the introduced fresh names in precise justifications can, and should, be eliminated for the purposes of presentation. Justifications with the fresh names removed are also referred to as "precise justifications".

In terms of splitting and flattening of axioms, the work by Baader et al.[BPS07] pinpoints axioms for entailments in the description logic $\mathcal{EL}$, and uses a normalisation procedure that produces flat small axioms. Although this normalisation procedure is really part of the $\mathcal{EL}$ classification algorithm, and the work is not concerned with fine-grained justifications, the basic idea could be used to indicate the parts of axioms that are responsible for the entailment.

Syntax Relevance, Sub-Concepts and Generalisation In [SC03], Schlobach and Cornet focus on computing justifications for unsatisfiable classes. They introduce the idea of *generalised terminologies*, which they describe as terminologies where some of the definitions have been generalised, and moreover, these generalisations are *syntactically related* to the original axioms. Schlobach and Cornet state that the notion of one definition being syntactically related to another definition needs to be specified for different knowledge representation languages and different types of knowledge bases, but they use an abstract notion of two *concepts* being syntactically related $\mathsf{rel}(C, C')$ to define syntactic generalisation of concepts as follows: A concept C' is a *syntactic generalisation of a concept* C if $\mathsf{rel}(C, C')$ and $C \sqsubseteq C'$ independent of a TBox. From this, they define a terminology $\mathcal{T}' = \{C_1 \sqsubseteq D'_1, \ldots, C_n \sqsubseteq D'_n\}$ to be a *generalised incoherence-preserving terminology* (GIT) of $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$ if each D'_i is a syntactic generalisation of D_i $(1 \leq i \leq n)$, and, every $\mathcal{T}'' = \{C_1 \sqsubseteq D''_1, \ldots, C_n \sqsubseteq D''_n\}$, where each D''_i is a syntactic generalisation of D'_i, is coherent. They define a generalisation strategy for unfoldable $\mathcal{ALC}$, where a concept C' is a generalisation of C if C' occurs as a subconcept in the concept that is obtained by unfolding of the axioms that define C, with polarity and quantifier depth preserved. Schlobach and Cornet coin the name *qualified sub-concepts* to describe generalised concepts of this type. These notions are best illustrated with an example. Consider the example from [SC03], where

$$\begin{aligned}\mathcal{O} = \{&A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3\\ &A_2 \sqsubseteq A \sqcap A_4\\ &A_3 \sqsubseteq A_4 \sqcap A_5\\ &A_4 \sqsubseteq \forall s.B \sqcap C\\ &A_5 \sqsubseteq \exists s.\neg B\\ &A_6 \sqsubseteq A_1 \sqcup \exists r.(A_3 \sqcap \neg C \sqcap A_4)\\ &A_7 \sqsubseteq A_4 \sqcap \exists s.\neg B\}\end{aligned}$$

According to their definition *some* of the qualified sub-concepts of A_1 are $\neg A \sqcap A \sqcap \forall s.B \sqcap C \sqcap \exists s.\neg B$ (which is obtained from the unfolding of the definition of A_1), $\neg A \sqcap A \sqcap \forall s.B$ (the conjunction of a subset of the conjuncts from the unfolding of A_1—all possible sets of conjuncts can be combined in this way to produce qualified sub-concepts) and $A \sqcap \exists s.\top$ (the concepts $\top$ and $\bot$ are in the set of qualified sub-concepts of any concept). Given the set MIPS of $\mathcal{O}$ (justifications for *root unsatisfiable classes*—see Section 2.3.2 on Page 46), the following generalised terminologies can be obtained.

$$\begin{aligned}\mathcal{O}'_1 &= \{A_1 \sqsubseteq \neg A \sqcap A\} && \text{(conjuncts from the unfolding of } A_1)\\ \mathcal{O}'_2 &= \{A_3 \sqsubseteq \forall s.B \sqcap \exists s.\neg B\} && \text{(conjuncts from the unfolding of } A_3)\\ \mathcal{O}'_3 &= \{A_7 \sqsubseteq \forall s.B \sqcap \exists s.\neg B\} && \text{(conjuncts from the unfolding of } A_7)\end{aligned}$$

Fine-grained Tableau Tracing As seen in Chapter 3, various researchers have used Baader and Hollunder's tableau tracing technique for computing justifications. In all of these approaches, tracing occurs at the level of axioms. However, in [LPSV06, Lam07, LSPV08], Lam extends the technique, and in addition to keeping track of which axioms are used to expand a completion $\mathcal{A}$-box, her technique also keeps track of the *sub-concepts* that are used in the expansion. Once a clash is detected, the labels can be read off the assertions in the completion ABox that contains the clash to reveal the parts of axioms that are responsible for the clash. These labels can then be translated to what Lam calls a "fine-grained justification". Given the way that a tableau algorithm breaks down axioms, and gradually processes sub-concepts, the obtained justifications are very similar to Kalyanpur's precise justifications. When it comes to relating the asserted axioms to the parts of the axioms that are responsible for a clash the details are a little sketchy. In particular, in common with Kalyanpur's definitions, it is not completely clear how to get from axioms in negation normal form back to the axioms in the original ontology.

Heuristic Based Techniques and Presentation The ontology editor Swoop features the ability to "strike out" the irrelevant parts of axioms from a justification. An example of strike out in action is shown in Figure 7.5. Although Kalyanpur, the primary author of Swoop, proposed the definition of precise justifications based on ontology splitting, as well as an algorithm to compute precise justifications based on splitting, no implementation of the algorithm exists. Instead, Swoop uses a heuristic approach. The heuristic that is used, is to strike out any name that only *occurs in one position* in the union of the entailment and justification. For example, given a justification

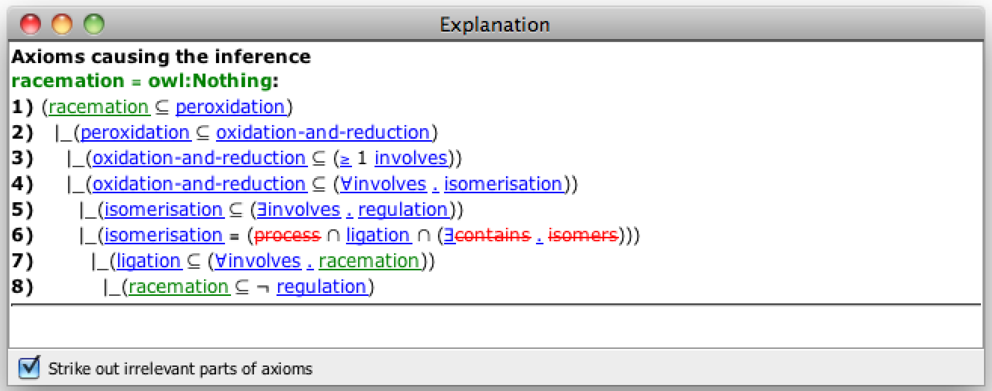
$$\mathcal{J} = \{A \sqsubseteq D \sqcap \exists R.C,\ \geq 1R \sqsubseteq B\}$$

for $A \sqsubseteq B$, the terms C and D would be struck out because they each only occur in one position in $\mathcal{J} \cup \{A \sqsubseteq B\}$. An advantage of this technique is that it is very efficient and easy to implement. The disadvantage of this technique is that it is obviously incomplete. For example, given

$$\mathcal{J} = A \sqsubseteq B \sqcap \exists R.C \sqcap \forall R.C \models A \sqsubseteq B$$

the technique would fail to eliminate the second and third conjuncts on the right hand side of the axiom, because both R and C appear more than once in $\mathcal{J} \cup \{A \sqsubseteq B\}$. One of the motivations for fine-grained justifications is that regular justifications are susceptible to the phenomena of masking. This thesis roughly characterises masking as the situation where the number of reasons for an entailment is not reflected by the number of justifications for that entailment. Masking was introduced by Kalyanpur in [KPG06], where he refers to "parts of axioms as being *masked*", and provides a gist of masking using the following example: Consider the ontology

$$\begin{aligned} \mathcal{O} = \{&A \sqcup C \sqsubseteq B \sqcap \exists R.D \sqcap E \sqcap \neg B \\ &A \sqsubseteq E \sqcap \forall R.\neg D\} \models A \sqsubseteq \bot \end{aligned}$$


Explanation

Axioms causing the inference
racemation ≡ owl:Nothing:

```
1) (racemation ⊆ peroxidation)
2)  |_(peroxidation ⊆ oxidation-and-reduction)
3)    |_(oxidation-and-reduction ⊆ (≥ 1 involves))
4)    |_(oxidation-and-reduction ⊆ (∀involves . isomerisation))
5)      |_(isomerisation ⊆ (∃involves . regulation))
6)      |_(isomerisation ≡ (~~process~~ ∩ ligation ∩ (∃~~contains~~ . ~~isomers~~)))
7)        |_(ligation ⊆ (∀involves . racemation))
8)          |_(racemation ⊆ ¬ regulation)
```

[✓ Strike out irrelevant parts of axioms]

Figure 7.5: An example of Strike Out in Swoop

which entails that A is unsatisfiable. There is just one justification for the unsatisfiability of A, namely

$$\mathcal{J} = \{A \sqcup C \sqsubseteq B \sqcap \exists R.D \sqcap E \sqcap \neg B\}$$

However, Kalyanpur points out that "additional parts of axioms that could contribute to the entailment are masked, e.g., the [the first and second axioms] can be broken down into $A \sqsubseteq \exists R.D$ and $A \sqsubseteq \forall R.\neg D$ which also entail the unsatisfiability of A, however, this condition cannot be captured [by justifications]." Although Kalyanpur coined the term masking, he did not provide any proper definitions of masking. He, also approached masking with a broad-brush view and, as can be seen by the example he provided, only really identified what this thesis now refers to as *external masking*.

Summary of Prior Approaches

A common aim of all previous approaches for computing fine-grained justifications is to determine the *parts* of axioms that are responsible for a particular entailment. However, none of these approaches define exactly what they mean by *parts* of axioms. Moreover, each approach is specific to a particular implementation technique and is defined in an *operational* sense. For example, Lam essentially uses the mechanics of a tableau algorithm to identify parts of axioms. This means that it is generally unclear as to what exactly constitutes a fine-grained justification. As a consequence, it is unclear as to whether any one approach for computing fine-grained justifications would result in the same set of fine-grained justifications for an entailment when compared to another approach.

While there are some arbitrary aspects of Kalyanpur's precise justifications, in particular, the choices made in defining the splitting function, and the work by Baader is not about fine-grained justifications per se, the main ideas of using some kind of normalisation and transformation that identifies sub-concepts are worth investigating.

Finally, the ideas behind Schlobach and Cornet's generalised TBoxes are interesting. They identify (1) the importance of syntax, presumably for the purposes of relating axioms in a GIT back to the axioms in the original terminology; (2) the most general form of a terminology in terms of entailment and preservation of incoherence, and (3) the idea of minimality with respect to the size of axioms and the number of concept names that appear in a GIT.

7.1.2 Other Notions of Redundancy and Minimality

In some seminal work [Qui59, Qui55, Qui52] Quine introduced the notion of a *prime implicant* of a formula, along with some other pertinent notions such as *redundancy*. In this work Quine was concerned with the problem of reducing a propositional formula to the shortest equivalent in a particular normal form, and to do this he used prime implicants.

Prime Implicants

According to Quine [Qui59], a *prime implicant* of a formula Φ is a *conjunction* of literals[1] that logically implies Φ but ceases to when deprived of any one literal. For example, consider the following formula, which is conjunctive normal form (CNF),

$$\psi = (p \vee \neg s) \wedge r \wedge (q \vee t).$$

Two prime implicants of ψ (there are more) are

$$F_1 = p \wedge r \wedge q \text{ and } F_2 = p \wedge r \wedge t$$

[1]Quine calls a conjunction of literals that does not have any repetitions a *fundamental formula*

On the other hand, the formula

$$F_3 = p \wedge q \wedge r \wedge t$$

is not a prime implicant of ψ, because although it entails ψ it is possible to drop the conjunct q with the result still entailing ψ. Although Quine talks about a single formula, it is of course possible to write a propositional theory into a single formula, so it also makes sense to talk about the prime implicants of a theory. In essence, the prime implicants of a formula Φ represent the *weakest* conjunctions of literals that entail Φ.

Finally, Quine also specifies the notion of a formula (in Conjunctive Normal Form) being *unilaterally redundant* if "it is equivalent to what remains of itself on dropping some one occurrence of a literal".

Prime Implicates

The dual notion of a prime implicant is a *prime implicate*. In propositional logic, a prime implicate of a formula Φ is a *disjunction* of literals that is entailed by Φ, but which is not entailed if deprived of any one literal. For example, consider again the formula

$$\psi = (p \vee \neg s) \wedge r \wedge (q \vee t)$$

then two prime implicates of ψ are

$$F_4 = r \text{ and } F_5 = q \vee t$$

However,

$$F_6 = q \vee t \vee r$$

is *not* a prime implicate, because although it is entailed by ψ, the literal r may be dropped from F_6 and the resulting formula is still entailed by ψ. In essence, prime implicates of a formula Φ represent the *strongest* disjunctions of literals that are entailed by Φ.

Prime Implicants and Prime Implicates for Description Logics

As seen above, prime implicants were originally defined for propositional logic, however, in [Bie09], Bienvenu defines notions of prime implicates for the multimodal logic $\mathbf{K}_n$ [BvW06][2]. Since $\mathcal{ALC}$ is a syntactic variant of $\mathbf{K}_n$ this also provides a definition of prime implicates and prime implicants for $\mathcal{ALC}$, and in [Bie08], Bienvenu presents work on a prime implicate normal form for $\mathcal{ALC}$ concepts. In her work, Bienvenu considers the properties of prime implicates for propositional logic, such as syntactic form (conjunctions of literals and disjunctions of literals), and the complexity of checking the satisfiability of prime implicates, and uses these qualities to select an appropriate definition for prime implicates for formulae in $\mathbf{K}_n$. Bienvenu uses well defined criteria to settle on a definition for the notions of literals, clauses (disjunctions of literals) and terms (conjunctions of literals) in $\mathbf{K}_n$ as being

$$\begin{aligned}
L &::= A \mid \neg A \mid \Box F \mid \Diamond F \\
C &::= L \mid L \vee L \\
T &::= L \mid T \wedge T \\
F &::= A \mid \neg A \mid F \wedge F \mid F \vee F \mid \Box F \mid \Diamond F
\end{aligned}$$

and then uses this definition to define a *clause* Φ as prime implicate of a formula ψ if $\psi \models \Phi$, and for any formula Φ', if $\Phi' \models \Phi$ then $\Phi \models \Phi'$. Intuitively, a prime implicate in $\mathbf{K}_n$ is the weakest clause that entails a given formula, where a clause has a specific structure and takes into consideration the modal operators in $\mathbf{K}_n$. Prime implicants are defined in the obvious way, as a strongest term (conjunction of literals) that entails a given formula. Again, a term has a specific structure.

Prime implicants and fine-grained justifications

At first sight, the various notions of prime implicants and prime implicates, such as redundancy, the notions of the weakest entailing formulae and the strongest entailed formulae of a given formula, are appealing when considering fine-grained justifications. This is due to the fact that, the notion of fine-grained justifications

[2]Bienvenu presents the work in the context of $\mathbf{K}_1$, and comments that the work is easily generalisable to $\mathbf{K}_n$

conjure up notions of redundancy and weakness. For example, given a justification

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

the conjunct C is superfluous, and the existential restriction $\exists R.D$ can be weakened so that, overall, the first axiom is weakened to $A \sqsubseteq \exists R.\top$ to give the fine-grained justification,

$$\mathcal{J}_1' = \{A \sqsubseteq \exists R.\top, \exists R.\top \sqsubseteq B\}$$

However, it is not immediately clear how a fine-grained justification relates to a prime implicant or a prime implicate. Given a justification $\mathcal{J} \models \eta$ and a fine-grained justification $\mathcal{J}'$ where $\mathcal{J} \models \mathcal{J}'$ and $\mathcal{J}' \models \eta$ (where $\mathcal{J}'$ is possibly weaker than $\mathcal{J}$ but possibly stronger than η), syntax issues aside, it may be the case that $\mathcal{J}'$ is a prime implicate of $\mathcal{J}$, and $\mathcal{J}'$ is a prime implicant of η, but in general this is obviously *not* the case. In the above example, syntax issues aside, $\mathcal{J}_1$ cannot be regarded as a prime implicant of $\mathcal{J}_1'$ since the conjunct B may be dropped from the first axiom and the result will still entail $\mathcal{J}_1'$.

In the context of fine-grained justifications, the notion of weakest entailing formulae (or sets of formulae) also needs to be approached with some caution. Consider the justification,

$$\mathcal{J}_2 = \{A \sqsubseteq \exists R.C \sqcap \forall R.C,\ \exists R.C \sqsubseteq B\} \models A \sqsubseteq B$$

Intuitively, there are two fine-grained justifications for the entailment $A \sqsubseteq B$, namely,

$$\mathcal{J}_2' = \{A \sqsubseteq \exists R.C,\ \exists R.C \sqsubseteq B\}$$

and

$$\mathcal{J}_2'' = \{A \sqsubseteq \exists R.\top \sqcap \forall R.C,\ \exists R.C \sqsubseteq B\}$$

It is obviously the case that $\mathcal{J}_2''$ is stronger than $\mathcal{J}_2'$, but it would be undesirable for either $\mathcal{J}_2'$ to rule out $\mathcal{J}_2''$ from being a fine-grained justification for $A \sqsubseteq B$, or vice-versa.

Finally, the syntax of prime implicants and prime implicates, in propositional logic and to some extent in Bienvenu's work, is very prescribed. Issues of syntax are obviously very important in both in the case of prime implicants/implicates and in the case of fine-grained justifications, but it seems clear that the syntax of prime implicates may be too far removed from the original axioms for them to be used in driving a definition of fine-grained justifications.

Overall, prime implicants and prime implicates bring several useful notions to the table, in particular, the notion of the weakest entailing formulae for a given formula, which may be useful for defining fine-grained justifications.

7.1.3 Fine-grained Repair

Finally, as mentioned earlier, one of the motivations in the literature for fine-grained justifications is fine-grained repair. In particular, the focus is on avoiding *over-repair*, where more entailments are lost "than is necessary". In the context of Description Logics and OWL, both Lam and Kalyanpur have investigated editing and removing parts of axioms in order to repair an ontology in a "minimal" way: In [Lam07, LSPV08] Lam discusses "minimal repairs", which are repairs that consist of removing parts of axioms so as to "lose as few atomic subsumptions as possible". In [Kal06] Kalyanpur mentions an *axiom rewrite module* in his repair tool, which is based on editing parts of axioms, to either strengthen or weaken them in order to repair an ontology. The main idea is that, "the module suggests a suitable rewrite of the axiom that preserves as much as information as possible while eliminating unsatisfiability.".

At the extreme end of the scale, Belief Revision [G9̈2] concerns the modification of a Belief Set (the deductive closure of a knowledge base, or *Belief Base*), to either add new knowledge or *revise* existing knowledge while keeping the knowledge base consistent. One of the key principles in this area is the notion of minimal change, so the the amount of "information lost" in a belief change is kept minimal [AGM85]. A more detailed comparison is beyond the scope of this thesis, but as far as belief revision and ontologies are concerned, a very readable discussion and presentation of the issues may be found in [LM05].

7.2 Intuitions and Desiderata

In summary, a *general definition of fine-grained justifications* is needed. Ideally, such a definition would not be tied to a particular DL. This definition would then permit the evaluation and comparison of algorithms for computing fine-grained or "precise" justifications and, it would make it possible to investigate the underlying problem in a thorough way. The overriding purpose of the next chapter is to provide such a definition. To summarise, there appear to be several desirable features that a definition for fine-grained justifications should satisfy. In particular, it should **help to identify**:

- **Axiom superfluity** Each axiom in a fine-grained justification should not contain superfluous parts (as in Figure 7.1). If the parts of an axiom in a justification can be edited so that the axiom itself is weakened, but the entailment in question still holds then the axiom contains superfluous parts.

- **Justification masking** Fine-grained justifications should make it possible to identify the different reasons for an entailment within a justification, over a set of justifications, or over and ontology. It should be possible to identify internally masked justifications (Figure 7.2), externally masked justifications (Figure 7.3) and shared cores (Figure 7.4). As explained previously, the number of *reasons* (masked justifications) may be different to the number of *regular justifications*.

- **Minimal repairs** One of the main motivations for fine-grained justifications is that "removing" parts, or sub-concepts, from axioms in an ontology can result in a more minimal repair than a repair which involves removing whole axioms from the self same ontology. Any definition should therefore make it possible to identify the parts of axioms that can be removed to enact a repair. In the context of this thesis, the notion of removing sub-concepts (or sub-terms) from an axiom is taken to be the replacement of sub-concepts, with either $\top$ or $\bot$ depending on the position and polarity of the sub-concept in question. The basic idea is to replace sub-concepts with $\top$ if the sub-concept has a positive polarity, and $\bot$ if the sub-concept has a negative polarity.

In the next chapter these desiderata are taken forward into the definitions of two types of fine-grained justification which this thesis call *laconic* justifications and *precise* justifications.

Chapter 8

Laconic and Precise Justifications

In what follows a definition of fine-grained justifications is proposed. Two types of fine-grained justification are defined: (1) *laconic justifications*, which intuitively are justifications whose axioms do not contain any superfluous parts for the target entailment; and (2) *precise* justifications, which intuitively are justifications that identify the parts of axioms that could be edited in order to produce a repair.

Before these kinds of justifications are properly defined it is necessary to introduce several conceptual and logical pieces of machinery. First, the notion of a *superfluous part* of an axiom in the context in the context of a justification is pinned down. Second, it is necessary to revisit Plaisted and Greenbaum's structural transformation [PG86], which is used as a tool for flattening and pulling axioms apart in the definition of laconic justifications. Third and finally, it is necessary to define the notion of syntactic isomorphism for axioms, as syntax also plays a part in the definition of laconic justifications. Once these pieces of machinery have been presented and discussed, laconic justifications are first defined, and then this definition is used to arrive at a definition of precise justifications. Much of what follows therefore focuses on laconic justifications.

8.1 Superfluity and Weakness

Laconic justifications centre around the notions of superfluity and weakness. Roughly speaking, a justification $\mathcal{J}$ for an entailment η is laconic if: (1) $\mathcal{J}$ does not contain any axioms that contain any sub-concept *occurrences* (i.e. sub-concepts at specific positions) that are superfluous for η, and (2) $\mathcal{J}$ does not contain any sub-concept occurrences that could be weakened while preserving η.

In the context of this thesis, superfluity and weakness are defined as follows:

Superfluity For a justification $\mathcal{J}$ for an entailment η, a subconcept in a *positive* position in some axiom in $\mathcal{J}$ is regarded as being superfluous if it can be substituted for $\top$ without breaking η, and a subconcept in a *negative* position in some axiom in $\mathcal{J}$ is superfluous if it can be substituted for $\bot$ without breaking η. For example, consider

$$\begin{aligned} \mathcal{O} = \{\alpha_1\colon & A \sqsubseteq C \sqcap D \\ \alpha_2\colon & E \sqcup C \sqsubseteq B\} \models A \sqsubseteq B \end{aligned}$$

A justification for $\mathcal{O} \models A \sqsubseteq B$ is $\mathcal{J} = \{\alpha_1, \alpha_2\}$. In this justification the subconcept D at $\alpha_1|_{2.2}$ is superfluous as its polarity is positive, and if it is substituted with $\top$, so that α_1 is replaced with $\alpha_1[\top]_{2.2} = A \sqsubseteq C \sqcap \top$, then it is still the case that $\mathcal{J} \models A \sqsubseteq B$. Similarly, the subconcept E at $\alpha_2|_{1.1}$ is superfluous as its polarity is negative, and if replaced with $\bot$, so that α_2 is replaced with $\alpha_2[\bot]_{1.1} = \bot \sqcup C \sqsubseteq B$, then it is still the case that $\mathcal{J} \models A \sqsubseteq B$. This thesis defines D at $\alpha_1|_{2.2}$ (or simply D in this case) to be $\top$-*Superfluous* in $\mathcal{J}$ for η, and E at $\alpha_2|1.1$ (or simply E in this case) to be $\bot$-*Superfluous* in $\mathcal{J}$ for η. These two notions of superfluity are made more precise by Definition 6.

Definition 6 ($\top\bot$-superfluity)**.** *Given a justification $\mathcal{J} \models \eta$, an axiom $\alpha \in \mathcal{J}$, an occurrence of a subconcept $\alpha|_p \neq \top$ in α with positive polarity is $\top$-Superfluous if $\mathcal{J} \setminus \{\alpha\} \cup \{\alpha[\top]_p\}$ is a justification for η. Similarly, an occurrence of a subconcept $\alpha|_p \neq \bot$ in α with negative polarity is $\bot$-Superfluous if $\mathcal{J} \setminus \{\alpha\} \cup \{\alpha[\bot]_p\}$ is a justification for η.*

It should be noted that, for a given justification for a given entailment, an occurrence of a subconcept may be superfluous in one context, but not superfluous in another context. This occurs when *internal masking* (Figure 7.2) is present within a justification, and the justification contains two or more reasons as to why it supports the target entailment.

Weakness For a justification $\mathcal{J}$ for an entailment η, if an occurrence of a subconcept C in some axiom in $\mathcal{J}$ is not $\top\bot$-superfluous, according to Definition 6, then it should be as weak as possible. That is, it should not be possible to alter C, and in doing so, weaken the axiom that contains C without breaking η. For example, consider

$$\begin{aligned} \mathcal{O} = \{&\alpha_1 \colon A \sqsubseteq \geq 3R.C \\ &\alpha_2 \colon \exists R.C \sqsubseteq B\} \models A \sqsubseteq B \end{aligned}$$

A justification for $\mathcal{O} \models A \sqsubseteq B$ is $\mathcal{J} = \{\alpha_1, \alpha_2\}$. While neither α_1 or α_2 contain occurrences of superfluous concepts, the subconcept $\geq 3R.C$ at $\alpha_1|_2$ can be weakened to $\geq 1R.C$, so that α_1 is replaced with the weaker axiom $A \sqsubseteq \geq 1R.C$, and it is still the case that $\mathcal{J} \models A \sqsubseteq B$. It should be noted that, in this context, what constitutes a valid weakening is rather prescribed. The exact criteria, and the reasons for them, will become clear when the definition of laconic justifications is laid down later in the thesis.

$\top\bot$-superfluity and Weakening Based Repair The above intuitions of superfluity and weakness lean towards the idea of breaking an entailment (repair) through the "elimination" and/or weakening of subconcepts, rather than the elimination of whole axioms. The basic idea is that, given a justification $\mathcal{J}$ for η, whose axioms do not contain any superfluous subconcepts, and whose axioms contain subconcepts that are as weak as possible, the replacement or weakening of any subconcept with either $\top$ or $\bot$, depending on the polarity of the subconcept, will break the entailment. Repair based on these ideas obviously offers the possibility of a more fine-grained approach than traditional justification based repair, and is described in more detail following the definitions of laconic and precise justifications.

8.2 δ–The Structural Transformation

As hinted at above, one of the most important requirements is the ability to identify the *occurrence* of a subconcept in some axiom—that is, the notion of a *subconcept with position*. The definition of laconic justifications requires this, and achieves it using a *structure preserving* transformation function δ, which removes the nesting of subconcepts, and produces axioms that are "small" and "flat". The transformation δ, which is presented below, is essentially the structural transformation which was originally described in Plaisted and Greenbaum

[PG86], and also introduced in [NW01] and [BEL01]. Plaisted and Greenbaum's original transformation [PG86] essentially introduces fresh predicate names in order to remove nesting in First Order Formulae. The idea of introducing fresh names was first introduced by Tseitin [Tse68] as a method of avoiding the exponential blow up that can occur when transforming propositional formulae into clausal normal form.

In terms of Description Logics, the transformation takes a set of axioms $\mathcal{S}$, and produces a different set of axioms $\mathcal{S}' = \delta(\mathcal{S})$ that, while not equivalent to $\mathcal{S}$ is equi-consistent and equi-satisfiable to $\mathcal{S}$. That is, $\mathcal{S}'$ is consistent if and only if $\mathcal{S}$ is consistent, and a concept C is satisfiable with respect to $\mathcal{S}'$ if and only if it is satisfiable with respect to $\mathcal{S}$. Moreover, any model $\mathcal{I}$ of $\delta(\mathcal{S})$ is also a model of $\mathcal{S}$, and any model of $\mathcal{S}$ can be *extended* into a model of $\delta(\mathcal{O})$ by appropriate interpretation of the symbols that are in $\mathcal{S}'$ but not in $\mathcal{S}$.

Rewrite rules that apply the transformation to a set of Description Logic axioms are given in [MSH07], [MSH09] and [Mot06]. However, these rules first rewrite general concept inclusion axioms of the form $C \sqsubseteq D$ into clausal form, i.e. $\top \sqsubseteq \mathsf{nnf}(\neg C \sqcup D)$, and then apply the transformation. The original transformation as presented by Plaisted and Greenbaum did not do this, and the transformation δ, presented below, is more in keeping with this original transformation.

In what follows a set of transformation rules that is used in the definition of δ is presented. The set of rules is *deterministic*. For a given ontology $\mathcal{O}$, the left hand side of each transformation rule matches an axiom in $\mathcal{O}$, and either replaces it with an axiom or a set of axioms that is defined by the right hand side of the rule.

Definition 7. *[$\mathcal{SHOIQ}$ Rewrite Rules for the Structural Transformation δ]In the rewrite rules below, A_C, A_D and A_{C_i} are* fresh concept names *that are* ***not*** *in the signature of $\mathcal{O}$. The concepts C, D and C_i $(1 \leq i \leq n)$ are either complex* *concept expressions, concept names that* ***are*** *in the signature of $\mathcal{O}$, $\top$ or $\bot$*

T1	$C \equiv D \rightsquigarrow \{C \sqsubseteq D,\ D \sqsubseteq C\}$
R1	$S \equiv R \rightsquigarrow \{S \sqsubseteq R, R \sqsubseteq S\}$
A1	$C(a) \rightsquigarrow \{\{a\} \sqsubseteq C\}$
A2	$R(a,b) \rightsquigarrow \{\{a\} \sqsubseteq \exists R.\{b\}\}$
G1	$C \sqsubseteq D \rightsquigarrow \{A_C \sqsubseteq A_D,\ C \sqsubseteq A_C,\ A_D \sqsubseteq D\}$
P1	$A_D \sqsubseteq C_1 \sqcap \cdots \sqcap C_n \rightsquigarrow \{A_D \sqsubseteq C_i \mid 1 \leq i \leq n, C_i \neq \top\}$
P2	$A_D \sqsubseteq C_1 \sqcup \cdots \sqcup C_n \rightsquigarrow \{A_{C_i} \sqsubseteq C_i \mid 1 \leq i \leq n\} \cup \{A_D \sqsubseteq A_{C_1} \sqcup \cdots \sqcup A_{C_n}\}$
P3	$A_D \sqsubseteq \neg C,\ C \neq \bot \rightsquigarrow \{A_D \sqsubseteq \neg A_C,\ C \sqsubseteq A_C\}$
P4	$A_D \sqsubseteq \exists R.C,\ C \neq \top \rightsquigarrow \{A_D \sqsubseteq \exists R.A_C,\ A_C \sqsubseteq C\}$
P5	$A_D \sqsubseteq \forall R.C,\ C \neq \top \rightsquigarrow \{A_D \sqsubseteq \forall R.A_C,\ A_C \sqsubseteq C\}$
P6	$A_D \sqsubseteq\ \geq nR.C,\ C \neq \top \rightsquigarrow \{A_D \sqsubseteq\ \geq nR.A_C,\ A_C \sqsubseteq C\}$
P7	$A_D \sqsubseteq\ \leq nR.C,\ C \neq \bot \rightsquigarrow \{A_D \sqsubseteq\ \leq nR.A_C,\ C \sqsubseteq A_C\}$
N1	$C_1 \sqcap \cdots \sqcap C_n \sqsubseteq A_D \rightsquigarrow \{C_i \sqsubseteq A_{C_i} \mid 1 \leq i \leq n\} \cup \{A_{C_1} \sqcap \cdots \sqcap A_{C_n} \sqsubseteq A_D\}$
N2	$C_1 \sqcup \cdots \sqcup C_n \sqsubseteq A_D \rightsquigarrow \{C_i \sqsubseteq A_D \mid 1 \leq i \leq n, C_i \neq \bot\}$
N3	$\neg C \sqsubseteq A_D,\ C \neq \top \rightsquigarrow \{\neg A_C \sqsubseteq A_D,\ A_C \sqsubseteq C\}$
N4	$\exists R.C \sqsubseteq A_D,\ C \neq \bot \rightsquigarrow \{\exists R.A_C \sqsubseteq A_D,\ C \sqsubseteq A_C\}$
N5	$\forall R.C \sqsubseteq A_D,\ C \neq \bot \rightsquigarrow \{\forall R.A_C \sqsubseteq A_D,\ C \sqsubseteq A_C\}$
N6	$\geq nR.C \sqsubseteq A_D,\ C \neq \bot \rightsquigarrow \{\geq nR.A_C \sqsubseteq A_D,\ C \sqsubseteq A_C\}$
N7	$\leq nR.C \sqsubseteq A_D,\ C \neq \top \rightsquigarrow \{\leq nR.A_C \sqsubseteq A_D,\ A_C \sqsubseteq C\}$

Rule T1 rewrites concept equivalence axioms into general concept inclusion axioms. Rule R1 rewrites role equivalence axioms into two role inclusion axioms. For the sake of convenience, rules A1 and A2 rewrite ABox concept and role assertions into TBox axioms using nominals. Rules G1, P1 – P7 and N1 – N7 rewrite general concept inclusion axioms into multiple axioms, flattening out all nested concept expressions. They do this by introducing fresh names for subconcepts and introducing "defining axioms" for these fresh names. Rules $\mathsf{P1} - \mathsf{P7}$ deal with positive occurrences of subconcepts. If a fresh concept name A_C is introduced for the concept C which is in a positive position then the defining axiom takes the form of $A_C \sqsubseteq C$. Rules $\mathsf{N1} - \mathsf{N7}$ deal with negative occurrences of subconcepts. If a fresh concept name A_C is introduced for the concept C which is in a negative position then the defining axiom takes the form of $C \sqsubseteq A_C$.

It should be noted that the rewrite rules do not introduce fresh concept names for nested positive occurrences $\top$ or nested negative occurrences of $\bot$. This helps to ensure termination. Additionally, direct use of $\top$ and $\bot$ is considered to be an instance of *trivial* superfluity (as opposed to the notion of superfluity defined in Definition 6) which is obvious and not detrimental to repair, for example, if $\mathcal{J} = \{A \sqsubseteq \exists R.\top, \exists R.\top \sqsubseteq B\} \models A \sqsubseteq B$ contains some kind of superfluity in terms of the filler $\top$ for the $\exists R.\top$ restriction—however, it is obvious that this filler is superfluous and that it cannot be replaced with $\top$ to repair the entailment $A \sqsubseteq B$. Finally, in comparison to the original structural transformation defined in [PG86], the transformation presented here goes one step further and ensures that nesting is removed from literal concepts such as $\neg A$. That is, it introduces a name for the subconcept occurrence of A.

Definition 8. *[The Structural Transformation δ for $\mathcal{SHOIQ}$] For a given set of $\mathcal{SHOIQ}$ axioms $\mathcal{O}$, $\mathcal{O}' = \delta(\mathcal{O})$ is the result of* exhaustively *applying the rewrite rules given in Definition 7.*

The following two lemmas capture the fact that δ preserves consistency, satisfiability and entailment.

Lemma 1. *For an ontology $\mathcal{O}$, any model $\mathcal{I}_\delta$ of $\delta(\mathcal{O})$ is also a model of $\mathcal{O}$, and any model $\mathcal{I}$ of $\mathcal{O}$ can be extended into a model $\mathcal{I}_\delta$ of $\delta(\mathcal{O})$ simply by interpreting the additional vocabulary introduced as part of the transformation δ.*

Proof. See the proof of Theorem 2 in [PG86]. □

Lemma 2. *If $\mathcal{J}$ is a justification for $\mathcal{O} \models \eta$, then there is a $\mathcal{J}_\delta \subseteq \delta(\mathcal{J})$ that is a justification for $\delta(\mathcal{J}) \models \eta$.*

Proof. It follows from Lemma 1 that $\delta(\mathcal{J}) \models \eta$. Hence, $\delta(\mathcal{J})$ must contain at least one subset minimal set of axioms that entails η—that is, $\delta(\mathcal{J})$ contains at least one justification for η with respect to $\delta(\mathcal{J})$. □

Structural Transformation Example Since the rewrite rules for the structural transformation δ are rather complex, the following example illustrates how it introduces fresh names for subconcepts and how these are used to remove subconcept nesting. Consider,

$$\mathcal{S} = \{\exists R.C \sqsubseteq D \sqcap \geq 2S.\neg E\}$$

applying the transformation to $\mathcal{S}$ gives:

$$\begin{aligned} \delta(\mathcal{S}) = \{X_0 &\sqsubseteq X_1 \\ \exists R.X_2 &\sqsubseteq X_0 \\ C &\sqsubseteq X_2 \\ X_1 &\sqsubseteq X_3 \\ X_1 &\sqsubseteq X_4 \\ X_3 &\sqsubseteq D \\ X_4 &\sqsubseteq \geq 2S.X_5 \\ X_5 &\sqsubseteq \neg X_6 \\ E &\sqsubseteq X_6\} \end{aligned}$$

where names of the form X_i represent fresh concept names that are introduced in order to remove concept nesting. The transformation begins with the introduction of the "root axiom" $X_0 \sqsubseteq X_1$, and the fresh concept names X_0 and X_1. In this case X_0 represents $\exists R.C$, and X_1 represents $D \sqcap \geq 2S.\neg E$. Next, the rewrite rules are recursively applied to these top level sub-concepts and so on, so that deeper nesting is removed in the same way. As can be seen, subconcepts with negative polarities are dealt with slightly differently to subconcepts with positive polarities. For example, the $\exists R.C$ subconcept on the left hand side of the original GCI, which is implicitly negated, is flattened out with the name X_0 and a defining axiom $\exists R.X_2 \sqsubseteq X_0$ is introduced (the introduced name appears on the right hand side of the introduced defining axiom). An example of a positive renaming is the introduction of the name X_3 in order to remove the nesting of the D subconcept on the right hand side of the GCI. In this case, the defining axiom $X_3 \sqsubseteq D$ is introduced (the introduced name appears on the left hand side of the defining axiom).

δ-Transformation Axiom Forms Applying the structural transformation δ to a set of axioms $\mathcal{S}$ results in a set of small flat axioms $\delta(\mathcal{S})$. Each axiom in $\delta(\mathcal{S})$ is of one of the forms shown in Definition 9, where the symbols N_*, P_*, and A represent concept names, R a $\mathcal{SHOIQ}$ role, o an individual name, and n a positive integer. Each concept name P_c represents a *positive occurrence* of some subconcept C in the original set of axioms, with an axiom of the form $P_c \sqsubseteq C$ "defining" that occurrence. Each concept name N_c represents a *negative occurrence* of some subconcept C in the original set of axioms, with each axiom of the form $C \sqsubseteq N_c$ "defining" that occurrence.

Definition 9 ($\mathcal{SHOIQ}$ δ-transformation axiom forms)**.** *For a set $\mathcal{S}$ of $\mathcal{SHOIQ}$ axioms, each axiom $\alpha \in \delta(\mathcal{S})$ must be one of the forms, where the symbols N_*, P_*, and A represent concept names, R a $\mathcal{SHOIQ}$ role, o an individual name, and n a positive integer:*

A1	$N_1 \sqsubseteq P_1$
P1	$P_c \sqsubseteq P_1 \sqcup \cdots \sqcup P_n$
P2	$P_c \sqsubseteq \neg N_1$
P3	$P_c \sqsubseteq \{o\}$
P4	$P_c \sqsubseteq A$
P5	$P_c \sqsubseteq \exists R.P_1$
P6	$P_c \sqsubseteq \forall R.P_1$
P7	$P_c \sqsubseteq \geq nR.P_1$
P8	$P_c \sqsubseteq \leq nR.N_1$
N1	$N_1 \sqcap \cdots \sqcap N_n \sqsubseteq N_c$
N2	$\neg N_1 \sqsubseteq N_c$
N3	$\{o\} \sqsubseteq N_c$
N4	$A \sqsubseteq N_c$
N5	$\exists R.N_1 \sqsubseteq N_c$
N6	$\forall R.N_1 \sqsubseteq N_c$
N7	$\geq nR.N_1 \sqsubseteq N_c$
N8	$\leq nR.P_1 \sqsubseteq N_c$
O1	$\mathsf{trans}(R)$

8.3 $\mathcal{SHOIQ}$ Syntactic Isomorphism

The final piece of machinery that is needed for the definition of laconic justifications is *syntactic isomorphism*. Intuitively, an axiom α' is syntactically isomorphic to another axiom α'' if there is an injective renaming ρ of the signature, with $\top$ and $\bot$, of α' so that $\rho(\alpha')$ is structurally equal to α''. For example, $\alpha' = A \sqsubseteq \exists R.B$ is isomorphic to $\alpha'' = F \sqsubseteq \exists S.B$, since A and R in α' can be renamed to F and S respectively, to make α' structurally equal to α''. Definition 10 captures what it means to be isomorphic for an axiom that occurs in the set of axioms $\delta(\mathcal{S})$ where $\mathcal{S}$ is a set of $\mathcal{SHOIQ}$ axioms. Note that structural equality is used as the ordering of conjuncts in a conjunction and disjuncts in a disjunction is unimportant here.

Definition 10 ($\mathcal{SHOIQ}$ δ-Isomorphism)**.** *Two $\mathcal{SHOIQ}$ axioms, α' and α'' are δ-isomorphic if α' and α'' are both of **one** of the forms given in Definition 9, and there is a injective renaming of each $N' \in \mathsf{signature}(\alpha') \cup \{\top, \bot\}$ into a name $N'' \in \mathsf{signature}(\alpha'') \cup \{\top, \bot\}$.*

When it is clear from the context (because the axioms being compared are contained in the result of applying the δ-transformation to a set of axioms), the prefix δ may be dropped, and the term *isomorphism* (resp. *isomorphic*) simply used to mean δ-isomorphism (resp. δ-isomorphic).

8.4 Laconic And Precise Justifications

With a suitable structural transformation, δ, in hand, laconic justifications can be defined. (Recall that $\mathcal{O}^\star$ is the deductive closure of $\mathcal{O}$)

Definition 11. *(**Laconic Justification**) Let $\mathcal{O}$ be an ontology such that $\mathcal{O} \models \eta$, $\mathcal{J}$ is a* laconic justification *for η over $\mathcal{O}$ if:*

1. *$\mathcal{J}$ is a justification for η in $\mathcal{O}^\star$*
2. *For every $\mathcal{J}_\delta \subsetneq \delta(\mathcal{J})$ it is the case that $\mathcal{J}_\delta \not\models \eta$*
3. *For each $\alpha \in \delta(\mathcal{J})$ there is no α' such that*
   - *(a) $\alpha \models \alpha'$ and $\alpha' \not\models \alpha$ (α' is weaker than α)*
   - *(b) α' is δ-isomorphic to α*
   - *(c) $\delta(\mathcal{J}) \setminus \{\alpha\} \cup \{\alpha'\}$ is a justification for η in $(\delta(\mathcal{O}))^\star$*

Intuitively, a laconic justification is a justification whose axioms do not have any superfluous parts (subconcepts) for the entailment in question, and, all of the non-superfluous parts are as weak as possible while still being of the same syntactic form.

Definition 12. *(**Precise Justification**) Let $\mathcal{O}$ be an ontology such that $\mathcal{O} \models \eta$. Let $\mathcal{J}$ be a justification for $\mathcal{O}^\star \models \eta$ and let $\mathcal{J}_p = \delta(\mathcal{J})$. $\mathcal{J}_p$ is* precise *with respect to $\mathcal{J}$ if $\mathcal{J}$ is a laconic justification for $\mathcal{O} \models \eta$.*

As can be seen, a precise justification is a laconic justification where all parts are exposed as axioms.

8.5 A Discussion of Definition 11

Use of Deductive Closures It is apparent from Definition 11(1) that the laconic justifications for $\mathcal{O} \models \eta$ may be drawn from the deductive closure of $\mathcal{O}$. Therefore, unlike regular justifications, laconic justifications are not specific to the asserted axioms in an ontology. This ensures that (1) it is possible to consider weaker parts of axioms when discarding superfluous parts, and that (2) it is possible to capture the phenomena of masking, which was discussed in the introduction to this chapter and is described in more detail in Chapter 9.

Constrained Axiom Weakening Definition 11(3a) captures the notion of subconcept weakening using isomorphism. It should be noted that in prior publication of this work (in [HPS08b]), this notion of isomorphic axiom weakening was captured by explicitly introducing the notion of axiom length into the definition of laconic justifications. With the more elegant tool of isomorphism axiom length no longer required.

Justifications for Tautologies are Laconic It follows from Definition 11 that justifications for tautologies are laconic: By Definition 1, a justification for a tautology is the empty set, which trivially satisfies Definition 11(1), Definition 11(2) and Definition 11(3).

8.6 Laconic Justifications Examples

Consider the following ontology

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B,\\ &B \sqsubseteq D,\\ &A \sqsubseteq B \sqcap C\} \models A \sqsubseteq D\end{aligned}$$

There are two justifications for $\mathcal{O} \models A \sqsubseteq D$, $\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq D\}$ and $\mathcal{J}_2 = \{B \sqsubseteq D, A \sqsubseteq B \sqcap C\}$. By Definition 11, $\mathcal{J}_1$ *is* a laconic justification since

$$\begin{aligned}\delta(\mathcal{J}_1) = \{&X_0 \sqsubseteq X_1,\\ &A \sqsubseteq X_0,\\ &X_1 \sqsubseteq B,\\ &X_2 \sqsubseteq X_3,\\ &B \sqsubseteq X_2,\\ &X_3 \sqsubseteq D\}\end{aligned}$$

which is a justification for η with respect to $\delta(\mathcal{J})$, moreover, none of these axioms can be weakened further without resulting in $\mathcal{J}_1 \not\models A \sqsubseteq D$. Conversely, $\mathcal{J}_2$ *is not* a laconic justification since

$$\begin{aligned}\delta(\mathcal{J}_2) = \{&X_0 \sqsubseteq X_1,\\ &B \sqsubseteq X_0,\\ &X_1 \sqsubseteq D\\ &X_2 \sqsubseteq X_3\\ &A \sqsubseteq X_2\\ &X_2 \sqsubseteq X_3 \sqcap X_4\\ &X_3 \sqsubseteq B\\ &X_4 \sqsubseteq C\}\end{aligned}$$

which is a superset of a justification for η is obtained, since $X_4 \sqsubseteq C$ is not required to entail η.

In essence, after applying the δ-transformation to a justification $\mathcal{J}$ for $\mathcal{O} \models \eta$ the resulting set of axioms $\delta(\mathcal{J})$ must entail η, but it may or may not be a justification for η with respect to itself. There are of course two options:

1. $\delta(\mathcal{J})$ *is* a justification for η—in this case, each axiom in $\delta(\mathcal{J})$ can be examined to make sure that it is as weak as possible, inline with the kinds of weakenings permitted by Definition 11, while still entailing η.

2. $\delta(\mathcal{J})$ is *not* a justification for η—in this case, $\delta(\mathcal{J})$ must be a superset of a justification for η, and the axioms in $\mathcal{J}$ must contain superfluous parts. Hence $\mathcal{J}$ cannot be laconic.

8.7 Key Properties of Laconic Justifications

The following Theorems summarise key properties of laconic justifications:

Theorem 1 (Number of Laconic Justifications). *Let $\mathcal{S}$ be a set of $\mathcal{SHOIQ}$ axioms such that $\mathcal{S} \models \eta$. There are an infinite number of laconic justifications over $\mathcal{S}$ for $\mathcal{S} \models \eta$.*

Proof. It follows from Definition 11 that, given a laconic justification $\mathcal{J} \models \eta$, any positive polarity class expression $C = \alpha|_p$ for some $\alpha \in \mathcal{J}$ may be substituted with $C \sqcap \top$, so that $\mathcal{J}' = \mathcal{J} \setminus \{\alpha\} \cup \{\alpha[C \sqcap \top]_p\}$, and the resulting justification

$\mathcal{J}'$ must be laconic. Since each substitution of this form produces an axiom that is in the deductive closure of the original axiom, and there can be an infinite number of such substitutions, it follows that there are an infinite number of laconic justifications for a given entailment. □

The proof of Theorem 1 uses "syntactic fluff" to show that there are an infinite number of laconic justifications for an entailment. Another construction that involves this kind of syntactic fluff is one based on the use of negation. Given a laconic justification $\mathcal{J}$, rather than substitute a positive polarity class expression C in some $\alpha \in \mathcal{J}$ with $C \sqcap \top$, C can be substituted with $\neg(\neg C)$. The resulting justification will still be laconic. This substitution can of course be performed ad infinitum without the loss of laconicity.

A less trivial example, which illustrates that there can be an infinite number of justifications for an entailment involves the use of number restrictions. Consider an ontology $\mathcal{O} \models A \sqsubseteq \bot$. It is possible to construct an infinite set of justifications for the unsatisfiability of A of the form $\{A \sqsubseteq \geq nR.\top, A \sqsubseteq \leq (n-1)R.\top\}$ $(1 < n)$. All of these axioms are in the deductive closure of $\mathcal{O}$.

Theorem 2. *For a given justification $\mathcal{J} \models \eta$, if $\mathcal{J}$ is laconic then it does not contain any $\top\bot$-superfluity.*

Proof. Assume for the moment that $\mathcal{J}$ *does* contain a $\top\bot$-superfluous subconcept $C = \alpha|_p$ for some $\alpha \in \mathcal{J}$. Since C is assumed to be superfluous, it must be the case that $C \neq \top$ and $C \neq \bot$ (Definition 6), and there must also be a "defining axiom" for C in $\delta(\mathcal{J})$ (Definition 7 and Definition 8). If C really is superfluous, it must be possible to substitute it with $\top$ or $\bot$ (depending on its polarity) in $\mathcal{J}$ and η must still be entailed by the result. Such a substitution amounts to removing the defining axiom for the superfluous subconcept from $\delta(\mathcal{J})$. Hence, if $\mathcal{J}$ contains superfluity, there must be some proper subset $\mathcal{J}' \subsetneq \delta(\mathcal{J})$ which entails η, and $\delta(\mathcal{J})$ cannot be a subset minimal set that entails η, thus violating Definition 11(3) meaning $\mathcal{J}$ cannot be laconic. □

Theorem 3 (Laconic Repair). *For a laconic justification $\mathcal{J} \models \eta$, any axiom $\alpha \in \mathcal{J}$, and any subconcept $C = \alpha|_p$ where the polarity of C is positive and $C \neq \top$, $\alpha[\top]_p$ is weaker than α, and $\mathcal{J} \setminus \{\alpha\} \cup \{\alpha[\top]_p\} \not\models \eta$. Similarly, for any subconcept $D = \alpha|_p$ where the polarity of D is negative and $D \neq \bot$, $\alpha[\bot]_p$ is weaker than α and $\mathcal{J} \setminus \{\alpha\} \cup \{\alpha[\bot]_p\} \not\models \eta$.*

Proof. It follows from Theorem 2 that $\mathcal{J}$ does not contain any $\top\bot$-superfluity. Therefore, it follows from Definition 6 that substituting C with $\top$, or substituting D with $\bot$ must break the entailment η and result in a repair. □

In essence, for a given justification $\mathcal{J} \models \eta$, Theorem 3 shows that it is possible to use laconic and precise justifications to determine which subconcept occurrences in $\mathcal{J}$ can be selected for substitution with either $\top$ or $\bot$ in order to guarantee a repair.

8.8 Discussion

Before concluding this chapter it is worth mentioning two issues. The first relates to how the definitions of laconic and precise justifications were arrived and how syntax strongly influenced the final outcome. The second relates to dealing with redundancy in justifications, which is subtly different from superfluity.

The Importance Of Syntax Before settling on the definition of laconic justifications presented in this thesis, various alternatives were considered. Initially, there was an overwhelming draw towards a "clean and pure" definition that was blind to syntax. Early iterations of the definition used axiom weakening but did not identify syntax and certainly not occurrences of subconcepts in axioms. However, this proved unworkable. In particular, the definition admitted axioms that could have complex "tautological junk" appended to subconcepts—weakening alone was not powerful enough to prevent this. Later attempts included the notion of *axiom length* to prevent axioms containing these additional tautological parts. It then became apparent that considering the length of axioms in there entirety was not enough. In the presence of self contradictions the definition collapsed. At this point the structural transformation became important for identifying occurrences of subconcepts and the importance of syntax became evident. As will be seen in the next chapter, syntax plays a crucial role when it comes to justification masking.

Dealing with Redundancy The definition of laconic justifications (and hence precise justifications) strikes a balance between practicality/implementability and dealing with all forms of superfluity and redundancy. In particular, it is based on the simple notion of $\top\bot$-superfluity and the mechanisms of fine-grained repair that are associated with this notion. While the definition caters for this notion of superfluity, it does *not* fully deal with the notion of *redundancy* as articulated by Quine [Qui52], whereby a subconcept would be considered redundant if it could be "cut out" of an axiom without loss of *information* (entailments). For example, consider the following justification for $A \sqsubseteq B$,

$$\mathcal{J} = \{A \sqsubseteq \exists R.C \sqcup \geq 1R.C, \quad \exists R.C \sqsubseteq B\} \models A \sqsubseteq B.$$

This justification is a laconic justification by Definition 11. Substituting any positive occurrence of a subconcept with $\top$ *will* break the entailment. However, it would also be possible to "cut out" one of the disjuncts, without losing any information, and obtain an equivalent justification for the entailment. In this sense, either $\exists R.C$ or $\geq 1R.C$ is redundant.

The example of redundancy presented here is rather simple so as to illustrate the issues between redundancy and superfluity, but as can be imagined, redundancy in a justification could be the result of complex interaction between nested parts of different axioms. Addressing this kind of redundancy is beyond the scope of this thesis, but it presents an interesting and challenging issue which should be investigated as part of future work

8.9 Conclusions

This chapter has presented laconic and precise justifications as a type of fine-grained justification. The notion of $\top\bot$-superfluity was introduced as a means of characterising the problem space, and laconic justifications were introduced as fine-grained justifications that do not contain any of this superfluity. In essence, the axioms in laconic justifications do not contain any superfluous parts, and every part is as weak as possible. Plaisted and Greenbaum's structural transformation, which sits outside of this work, plays a central role in the definition of laconic and precise justifications and is used to identify each subconcept occurrence in an axiom. Finally, the logic toolbox containing the Plaisted and Greenbaum's structural transformation and syntactic isomorphism, that was used to define laconic and precise justifications is quite general and not specific to $\mathcal{SHOIQ}$. This should mean that it is possible to extend the definition to more expressive logics in the future.

Chapter 9

Justification Masking

As discussed in Chapter 7, one of the main motivations for fine-grained justifications is that regular justifications can mask further multiple reasons for an entailment. Despite the fact that the basic intuitions behind masking are easy to grasp, there have been no proper prior definitions or analysis of masking. In what follows, the phenomena of masking is examined in more detail.

9.1 Types of Masking

This thesis defines four important types of masking: *Internal Masking*, *Cross Masking*, *External Masking* and *Shared Core Masking.* The intuitions behind these types of masking are explained below.

Internal Masking

Internal masking refers to the phenomena where there are multiple reasons within a single justification as to why the entailment in question holds. An example of internal masking is shown below.

$$\mathcal{O} = \{A \sqsubseteq B \sqcap \neg B \sqcap C \sqcap \neg C\} \models A \sqsubseteq \bot$$

There is a single regular justification for $\mathcal{O} \models A \sqsubseteq \bot$, namely $\mathcal{O}$ itself. However, within this justification there are, intuitively, two reasons as to why $\mathcal{O} \models A \sqsubseteq \bot$, the first being $\{A \sqsubseteq B \sqcap \neg B\}$ and the second being $\{A \sqsubseteq C \sqcap \neg C\}$.

Cross Masking

Intuitively, cross masking is present within a set of justifications for an entailment when parts of axioms from one justification combine with parts of axioms from another justification in the set to produce new reasons for the given entailment. For example, consider the following ontology.

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B \sqcap \neg B \sqcap C\\ &A \sqsubseteq D \sqcap \neg D \sqcap \neg C\} \models A \sqsubseteq \bot\end{aligned}$$

There are two justifications for $\mathcal{O} \models A \sqsubseteq \bot$, namely $\mathcal{J}_1 = \{A \sqsubseteq B \sqcap \neg B \sqcap C\}$ and $\mathcal{J}_2 = \{A \sqsubseteq D \sqcap \neg D \sqcap \neg C\}$. However, part of the axiom in $\mathcal{J}_1$, namely $A \sqsubseteq C$ may combine with part of the axiom in $\mathcal{J}_2$, namely $A \sqsubseteq \neg C$ to produce a further reason: $\mathcal{J}_3 = \{A \sqsubseteq C, A \sqsubseteq \neg C\}$.

External Masking

While internal masking and cross masking take place over a set of "regular" justifications for an entailment, external masking involves parts of axioms from a regular justification combining with parts of axioms from an ontology (intuitively the axioms outside of the set of regular justifications) to produce further reasons for the entailment in question. Consider the example below,

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B \sqcap \neg B \sqcap C\\ &A \sqsubseteq \neg C\} \models A \sqsubseteq \bot\end{aligned}$$

There is just one justification for $\mathcal{O} \models A \sqsubseteq \bot$, however, although $A \sqsubseteq \neg C$ intuitively plays a part in the unsatisfiability of A it will never appear in a justification for $\mathcal{O} \models A \sqsubseteq \bot$. When $\mathcal{O}$ is taken into consideration, there are two salient reasons for $A \sqsubseteq \bot$, the first being $\{A \sqsubseteq B \sqcap \neg B\}$ and the second being $\{A \sqsubseteq C, A \sqsubseteq \neg C\}$

Shared Core Masking

Finally, two justifications share a *core* if after stripping away the superfluous parts of axioms in each justification the justifications are essentially structurally equal. Consider the example below,

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B \sqcap \neg B \sqcap C\\ &A \sqsubseteq B \sqcap \neg B\} \models A \sqsubseteq \bot\end{aligned}$$

There are two justifications for $\mathcal{O} \models \eta$, $\mathcal{J}_1 = \{A \sqsubseteq B \sqcap \neg B \sqcap C\}$ and $\mathcal{J}_2 = \{A \sqsubseteq B \sqcap \neg B\}$. However, $\mathcal{J}_1$ can be reduced to the laconic justification $\{A \sqsubseteq B \sqcap \neg B\}$ (since C is irrelevant for the entailment), which is structurally equal to $\mathcal{J}_2$. With regular justifications, it appears that there are more reasons for the entailment, when in fact each justification boils down to the same reason.

Masking Due to Weakening

The above intuitions have been illustrated using simple propositional examples. However, it is important to realise that masking is not just concerned with Boolean parts of axioms. *Weakest parts* of axioms must also be taken into consideration. For example, consider

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq \geq 2R.C\\ &A \sqsubseteq \geq 1R.D\\ &C \sqsubseteq \neg D\} \models A \sqsubseteq \geq 2R\end{aligned}$$

There is one regular justification for $\mathcal{O} \models A \sqsubseteq \geq 2R$ namely, $\mathcal{J}_1 = \{A \sqsubseteq \geq 2R.C\}$. However, there are intuitively two reasons for this entailment. The first is described by the justification obtained as a weakening of $\mathcal{J}_1$, and is $\mathcal{J}_2 = \{A \sqsubseteq \geq 2R\}$. The second is obtained by weakening the first axiom in $\mathcal{O}$ and combining it with the second and third axioms in $\mathcal{O}$ to give $\{A \sqsubseteq \geq 1R.C, A \sqsubseteq \geq 1R.D, C \sqsubseteq \neg D\}$.

Of course, masking due to weakening can occur in internal masking, cross masking, external masking and shared cores.

9.2 Summary on Masking Intuitions

As can be seen from the above examples, the *basic idea* is that when the weakest parts of axioms in a justification, set of justifications or an ontology are taken into consideration, there can be multiple reasons for an entailment that are otherwise not exposed with regular justifications. These reasons take the form of laconic justifications—justifications whose axioms do not contain any superfluous parts and whose parts are as weak as possible. With internal masking, cross masking and external masking, there are more laconic justifications (by some measure) than there are regular justifications. With shared cores there are fewer laconic justifications (by some measure) than there are regular justifications.

9.3 Detecting Masking

Given the above link between masking, weakest parts of axioms and laconic justifications, it may seem fruitful to use laconic justifications as a mechanism for detecting masking. Specifically, it may seem like a good idea to count laconic justifications for the entailment in question. However, this is a flawed intuition and several problems prevent laconic justification counting being used *directly* as a masking detection mechanism. The problems are outlined below.

The Promiscuity of the Deductive Closure

The first problem is that, in general, there can be an infinite number of laconic justifications for a given entailment (Theorem 1). The notion of counting the number of laconic justifications over a set of axioms and comparing this to the number of regular justifications over the same set of axioms is therefore useless when it comes to detecting and defining masking. Even if the logic used did not result in an infinite number of laconic justifications, the effects of splitting and syntactic equivalence could result in miscounting. For example, consider $\mathcal{J}_1 = \{A \sqsubseteq B \sqcap C, B \sqcap C \sqsubseteq D\}$, where $\mathcal{J}_1$ is in itself laconic, however another justification $\mathcal{J}_2 = \{A \sqsubseteq B, A \sqsubseteq C, B \sqcap C \sqsubseteq D\}$ can be obtained, which is also laconic. Clearly, masking is not present in $\mathcal{J}_1$, but there are more laconic justifications than there are regular justifications.

Preferred Laconic Justifications

Another approach might be to count the number of *preferred laconic justifications*,[1] which are a finite subset of laconic justifications that are made up of

[1]Preferred laconic justifications are discussed in more detail in Chapter 10

axioms which come from a filter on the deductive closure of a set of axioms. Unfortunately, this idea is sensitive to the definition of the filter. Different filters, for different applications, may give different answers and false positives. While a particular filter could be verified to behave correctly and perhaps be used as an optimisation for detecting masking in an implementation, this mechanism is not appropriate for defining masking.

Preservation of Positional Information

Another problem is that structural information is not directly preserved with laconic justifications. Consider $\mathcal{J} = \{\alpha \colon A \sqsubseteq B \sqcap (C \sqcap B)\}$ as a justification for $A \sqsubseteq B$. Masking is present within this justification. If $B_{@1} = \alpha|_{2.1}$ denotes the first occurrence of B, and $B_{@2} = \alpha|_{2.2.2}$ denotes the second occurrence of B then A is a subclass of B because of two reasons: $A \sqsubseteq B_{@1}$ and $A \sqsubseteq B_{@2}$. However, this positional information is lost in all laconic justifications for $A \sqsubseteq B$. In essence, syntax of asserted axioms is crucial when it comes to masking.

Splitting is Not Enough

While syntax is very important when considering masking, it does not suffice to consider syntax alone. The example of masking due to weakening shows that simply splitting a set of axioms $\mathcal{S}$ into their constituent parts, using the structural transformation $\delta(\mathcal{S})$, and then examining the justifications for the entailment with respect $\delta(\mathcal{S})$ is not enough to capture this notion of masking. Weakenings of the split axioms must be considered in any mechanism that is used to detect masking.

9.4 Masking Defined

With the above intuitions and desiderata in mind the notion of masking can be made more concrete. In the spirit of laconic justifications, the basic idea is to pull apart the axioms in a justification, set of justifications and an ontology, compute constrained weakenings of these parts (inline with the definition of laconic justifications), and then to check for the presence and number of laconic justifications within the set of regular justifications for an entailment with respect to these parts and their weakenings.

9.4.1 Parts and Their Weakenings

First, it is necessary define a function $\delta^+(\mathcal{S})$, which maps a set of axioms $\mathcal{S}$ to a set of axioms composed from the union of $\delta(\mathcal{S})$ with the constrained weakenings of axioms in $\delta(\mathcal{S})$. The weakenings of axioms are constrained in accordance with Definition 11(3). For an axiom $\alpha \in \delta(\mathcal{S})$, a weakening α' of α is contained in $\delta^+(\mathcal{S})$ only if α' is of the same form as α—i.e. α' is δ-isomorphic to α.

Definition 13 (δ^+). *For a set of $\mathcal{SHOIQ}$ axioms, $\mathcal{S}$,*

$$\begin{aligned}\delta^+(\mathcal{S}) := \delta(\mathcal{S}) \cup \{\alpha \mid \exists \alpha' \in \delta(\mathcal{S}) \ &s.t.\\ &\alpha' \models \alpha \ and\\ &\alpha \not\models \alpha' \ and\\ &\alpha' \ is \ \ isomorphic \ to \ \alpha\}\end{aligned}$$

Lemma 3 (δ^+justificatory finiteness). *For a finite set of axioms $\mathcal{S}$, the set of justifications for an entailment in $\delta^+(\mathcal{S})$ is finite.*

Proof. $\delta^+(S)$ is composed of the set of axioms in $\delta(\mathcal{S})$, which is finite, plus a possibly infinite set of axioms taken from the deductive closure of *each axiom* in $\delta(\mathcal{S})$. For a $\mathcal{SHOIQ}$ axiom α, every axiom α' in $\delta(\alpha)$ must be one of the following forms presented in Definition 9 (Page 147). Disregarding axioms in $\delta(\mathcal{S})$ of form P8 ($P_c \sqsubseteq \leq nR.N_1$) and N7 ($\geq nR.N_1 \sqsubseteq N_c$), the remaining set of axioms in $\delta^+(\alpha)$ is finite since the set of weakenings (in accordance with the definition of δ^+) of α' is finite. For axiom forms P8 and N7, α' is of the form: $X_i \sqsubseteq \leq nR.X_j$, in which case there is an *infinite* number of weakenings of α' in $\delta^+(\alpha)$ since $A \sqsubseteq \leq (n+1)R.C$ is weaker than $A \sqsubseteq \leq nR.C$ for any $n \geq 0$. If justifications are made up solely of the axioms of the form corresponding to the first set then the set of justifications is clearly finite. If justifications contain axioms of the second form $X_i \sqsubseteq \leq nR.X_j$ then there is a finite upper bound m for n, where there are no justifications containing an axiom of the from $X_i \sqsubseteq \leq kR.X_j$ for some $k > m$. This is because, for values of k, where k is equal to the maximum number in $\leq$ restrictions in the closure of $\mathcal{S}$, or more, $X_i \sqsubseteq \leq kR.X_j$ is too weak to participate in a justification, and this follows as a straight forward consequence of $\mathcal{SHOIQ}$'s model theory. □

Next, a function which filters out laconic justifications for an entailment from a set of all justifications for the entailment is defined in Definition 14.

Definition 14 (Laconic Filtering)**.** *For a set of axioms $\mathcal{S} \models \eta$, $\mathsf{laconic}(\mathcal{S}, \eta)$ is the set of justifications for $\mathcal{S} \models \eta$ that are laconic over $\mathcal{S}$.*

It should be noted that because of Lemma 3, the set of justifications $\mathsf{laconic}(\mathcal{S}, \eta)$ is finite.

9.4.2 Masking Definitions

With the definition of δ^+ and the definition of laconic filtering in hand, the various types of masking can now be defined.

Definition 15 (Internal Masking)**.** *For a justification $\mathcal{J}$ for $\mathcal{O} \models \eta$, internal masking is present within $\mathcal{J}$ if*

$$\left|\mathsf{laconic}(\delta^+(\mathcal{J}), \eta)\right| > 1$$

Theorem 4. *Internal masking is not present within a laconic justification.*

Proof. Assume that $\mathcal{J}$ is a laconic justification for η and that internal masking is present within $\mathcal{J}$. This means that there either must be (i) at least two laconic justifications for $\delta^+(\mathcal{J}) \models \eta$, i.e. there exists some $\mathcal{J}_1, \mathcal{J}_2 \subsetneq \delta^+(\mathcal{J})$ where $\mathcal{J}_1 \neq \mathcal{J}_2$ and are both laconic. However, since $\mathcal{J}$ itself is laconic this violates condition 2 of Definition 11, or (ii) there is an isomorphic weakening of one or more axioms in $\delta(\mathcal{J})$ that yields $\delta(\mathcal{J})'$. However since $\mathcal{J}$ is laconic this violates conditions 3a and 3b of Definition 11. □

Let $\mathcal{O} \models \eta$ and $\mathcal{J}_1, \ldots, \mathcal{J}_n$ be the set of all justifications for $\mathcal{O} \models \eta$. Cross masking and External masking are then defined as follows:

Definition 16 (Cross Masking)**.** *For two justifications $\mathcal{J}_i$ and $\mathcal{J}_j$, cross masking is present within $\mathcal{J}_i$ and $\mathcal{J}_j$ if*

$$\left|\mathsf{laconic}\big(\delta^+(\mathcal{J}_i \cup \mathcal{J}_j), \eta\big)\right| > \Big(\left|\mathsf{laconic}\big(\delta^+(\mathcal{J}_i), \eta\big)\right| + \left|\mathsf{laconic}\big(\delta^+(\mathcal{J}_j), \eta\big)\right|\Big)$$

Definition 17 (External Masking)**.** *External masking is present if*

$$\left|\mathsf{laconic}(\delta^+(\mathcal{O}), \eta)\right| > \left|\mathsf{laconic}(\delta^+(\bigcup_{i=1}^{i=n} \mathcal{J}_i), \eta)\right|$$

Definition 18 (Shared Cores)**.** *Two justifications $\mathcal{J}_i$ and $\mathcal{J}_j$ for $\mathcal{O} \models \eta$, share a core if there is a justification $\mathcal{J}_i' \in \mathsf{laconic}(\delta^+(\mathcal{J}_i), \eta)$ and a justification $\mathcal{J}_j' \in \mathsf{laconic}(\delta^+(\mathcal{J}_j), \eta)$ and a renaming ρ of terms not in $\mathcal{O}$ such that $\rho(\mathcal{J}_i') = \mathcal{J}_j'$.*

9.5 Conclusions

This chapter has presented issues related to detecting and defining justification masking. Four specific types of masking have been introduced and defined: Internal masking, Cross-masking, External masking, and Shared-Core masking. This is a somewhat more detailed breakdown of masking as identified in the literature, where only external masking was informally identified.

In summary, masking is a phenomena that results in the number of justifications for an entailment not reflecting the *number of reasons* for an entailment. The above definitions of masking basically identify the parts of axioms in a justification, over a set of justifications or over an ontology, weaken these parts and then look for the number of laconic justifications that are present in the set of justifications over the axioms that represent these weakened parts. As seen, syntax is crucial when it comes to masking. This is because it directly depends upon what has been written down into axioms. In addition to paying homage to syntax it is also necessary to consider semantics, and in particular the effect of weakening parts of axioms, as this makes it possible to capture masking due to weakening of cardinality restrictions in $\mathcal{SHOIQ}$. Finally, masking was one of the motivations for fine-grained justifications and, as will be seen later, masking is a natural phenomenon that occurs in realistic ontologies.

Chapter 10

Laconic Justification Finding Algorithms

With the definitions of laconic justifications and the various types of masking in hand, it is now possible to focus on detecting and computing laconic justifications for an entailments in ontologies. In what follows, a decision procedure for determining if a $\mathcal{SHOIQ}$ justification is laconic is presented. Next, the focus is shifted from detecting to finding laconic justifications. The notion of *preferred* laconic justifications is introduced and algorithms for computing these kinds of justifications are presented.

10.1 Detecting Laconic Justifications

Algorithm 10.1 is a decision procedure for determining whether or not a $\mathcal{SHOIQ}$ justification is laconic. The decision procedure answers "yes" if a justification $\mathcal{J} \models \eta$ is laconic for η, and answers "no" if $\mathcal{J}$ is not laconic for η. The algorithm has one sub-routine, ComputeDelta, which, given a set of axioms, computes the δ-transformation of the set of axioms.

10.1.1 The IsLaconic Algorithm

The algorithm works by first deriving the set of axioms S_δ by applying the δ-transformation to the input justification $\mathcal{J}$. If S_δ is not a justification for η with respect to itself then the input justification $\mathcal{J}$ must not be laconic, and the algorithm returns false. This corresponds to Definition 11(3). Next, each axiom

$\alpha \in S_\delta$ is examined. If α is an axiom that has a cardinality restriction on the left or right hand side, then a weaker version α' of α is obtained by either increasing or decreasing the cardinality, depending on the type of cardinality restriction and its polarity, by one. If α' can be substituted for α in S_δ with the resulting set still entailing η then $\mathcal{J}$ must not have been laconic and the algorithm returns false, otherwise $\mathcal{J}$ must be laconic and the algorithm returns true.

It is noticeable that Algorithm 10.1 does not check to see if axioms, whose left hand sides or right hand sides are not cardinality restrictions, can be weakened while preserving η. For example, given an axiom of the form $A \sqsubseteq \exists R.B$, the algorithm does not check to see if this axiom can be weakened to $A \sqsubseteq \exists R.\top$ while preserving the entailment. Neither does the algorithm check to see if the fillers of cardinality restrictions could be weakened to $\top$ or $\bot$ depending on polarity. The reason for this is that the transformation δ exposes the fillers of cardinality restrictions and other types of restrictions as extra axioms. If these fillers are superfluous, because they could be weakened in $\mathcal{J}$, then $\delta(\mathcal{J})$ would not be a justification for η, and the algorithm would return false at line 4.

Lemma 4 (Algorithm 10.1 Termination)**.** *For an input justification $\mathcal{J}$ and an entailment η, where $\mathcal{J} \models \eta$, Algorithm 10.1 terminates.*

Proof. In the worst case, the algorithm requires a single pass over the axioms in S_δ, with a maximum of two entailment checks per axiom. Given that $\mathcal{J}$ is finite, $\delta(\mathcal{J})$ is also finite. Hence, Algorithm 10.1 terminates on input $\mathcal{J}$ and η. □

Lemma 5 (Algorithm 10.1 Soundness and Completeness)**.** *For an input justification $\mathcal{J}$ and an entailment η, where $\mathcal{J} \models \eta$, Algorithm 10.1 returns* true *if and only if $\mathcal{J}$ is laconic for η, and returns* false *if and only if $\mathcal{J}$ is not laconic for η.*

Proof. According to Lemma 4, the algorithm terminates and so it must return either true or false. If $\mathcal{J}$ is empty (η must be a tautology), $\mathcal{J}$ is trivially laconic. In this case, $S_\delta(\mathcal{J})$ is the empty set, and the loop between lines 2 and 30 never gets executed, hence the algorithm returns true. If $\mathcal{J}$ is not empty then either $\mathcal{J}$ is laconic or $\mathcal{J}$ is not laconic. If $\mathcal{J}$ is *not* laconic then, by definition, either: (1) It is the case that there must be some *proper* subset of $\delta(\mathcal{J})$ that entails η. In this case, in lines 3–5, the algorithm will find some axiom $\alpha \in \delta(\mathcal{J})$ such that $\delta(\mathcal{J}) \setminus \{\alpha\} \models \eta$ and it will return false. (2) It is the case that there is some axiom $\alpha \in \delta(\mathcal{J})$ that can be weakened to α' in accordance with Definition 11(3), so that $\delta(\mathcal{J}) \setminus \{\alpha\} \cup \{\alpha'\}$ is a justification for η. In this case, α *must* be of one of

Algorithm 10.1 IsLaconic($\mathcal{J}$, η)

```
Require: $\mathcal{J} \models \eta$
1: $S_\delta \leftarrow$ ComputeDelta($\mathcal{J}$)
2: for $\alpha \in S_\delta$ do
3:     if $S_\delta \setminus \{\alpha\} \models \eta$ then
4:         return false
5:     end if
6:     if $\alpha$ is of the form $A \sqsubseteq \geq nR.B$ then
7:         $\alpha' \leftarrow A \sqsubseteq \geq (n-1)R.B$
8:         if $(S_\delta \setminus \{\alpha\}) \cup \{\alpha'\} \models \eta$ then
9:             return false
10:        end if
11:    end if
12:    if $\alpha$ is of the form $\geq nR.B \sqsubseteq A$ then
13:        $\alpha' \leftarrow \geq (n+1)R.B \sqsubseteq A$
14:        if $(S_\delta \setminus \{\alpha\}) \cup \{\alpha'\} \models \eta$ then
15:            return false
16:        end if
17:    end if
18:    if $\alpha$ is of the form $A \sqsubseteq \leq nR.B$ then
19:        $\alpha' \leftarrow A \sqsubseteq \leq (n+1)R.B$
20:        if $(S_\delta \setminus \{\alpha\}) \cup \{\alpha'\} \models \eta$ then
21:            return false
22:        end if
23:    end if
24:    if $\alpha$ is of the form $\leq nR.B \sqsubseteq A$ then
25:        $\alpha' \leftarrow \leq (n-1)R.B \sqsubseteq A$
26:        if $(S_\delta \setminus \{\alpha\}) \cup \{\alpha'\} \models \eta$ then
27:            return false
28:        end if
29:    end if
30: end for
31: return true
```

the following forms, $A \sqsubseteq \geq nR.B$, $\geq nR.B \sqsubseteq A$, $A \sqsubseteq \leq nR.B$ or $\leq nR.B \sqsubseteq A$. All of these forms are examined in lines 6–29, and if an axiom of one of these forms can be weakened by altering the cardinality by unity ($n - 1$ or $n + 1$ as appropriate) so that the entailment still holds then the algorithm returns false. If $\mathcal{J}$ *is* laconic, then none of the subset checks (lines 3–5) or weakening checks (lines 6–29) results in the η still holding and the point of execution arrives at line 31 and returns true. □

10.2 Preferred Laconic Justifications

Since regular justifications are subsets of ontologies, there is a finite number of them for any given entailment in any given ontology. However, as is evidenced by Theorem 1, the same is not true of laconic justifications. For any given entailment there can be an *infinite* number of laconic justifications. The question of which laconic justifications should be computed for a given entailment therefore arises. Going back to the original motivation for laconic justifications, there are two main ideas related to *understanding* and *repair* that suggest an answer to this question, and ultimately lead to the notion of *preferred laconic justifications.*

In essence, for a given ontology and entailment, the set of preferred laconic justifications should make it easy to understand how parts of axioms in *that ontology* give rise to the entailment, and also make it easy to devise a repair plan based on parts of axioms (that are exposed by precise justifications). In particular, the set of preferred laconic justifications should be:

- **Finite**—It should be possible to compute and examine each justification in the set of preferred laconic justifications. This is essential for the purposes of devising an automated repair, and to some extent is necessary for the purposes of understanding.

- **Syntactically relevant**—the axioms in preferred laconic justifications should somehow be syntactically related to the axioms in the ontology whose deductive closure they are drawn from. For example, consider the ontology

$$\begin{aligned} \mathcal{O} = \{&C \equiv D \sqcap \neg D \sqcap E, \\ &A \sqsubseteq B\} \end{aligned}$$

laconic justifications for $\mathcal{O} \models C \sqsubseteq \bot$ include (amongst others)

$$\mathcal{J}_1 = \{C \sqsubseteq D \sqcap \neg D\}$$

and

$$\mathcal{J}_2 = \{C \sqsubseteq D, C \sqsubseteq \neg D\}$$

It is noticeable that both of these justifications are somewhat structurally or syntactically related to the asserted axioms in $\mathcal{O}$ and both would be acceptable forms of preferred laconic justification for this entailment. However, there are other laconic justifications such as

$$\mathcal{J}_3 = \{C \sqsubseteq B \sqcap \neg B\}$$

that are *not structurally related* to axioms in $\mathcal{O}$. Despite the fact that $\mathcal{J}_3$ is a laconic justification for $C \sqsubseteq \bot$, it is arguable that justifications of this ilk, that could be considered to be syntactically irrelevant or “incidental”, are not of general interest to an ontology modeller who is trying to understand the reasons for an entailment. Hence, syntactic relevance is essential for usability and understanding.

- **Reason complete**—the set of preferred laconic justifications for an entailment should reflect all of the *reasons*, in terms of the *parts of axioms from the original ontology* in which the entailment holds. This idea is directly related to the notion of masking. Recall that, intuitively, masking arises when the number of regular justifications does not reflect the number of *reasons* for an entailment over a justification, set of justifications or an ontology. The notion of *reasons* for an entailment can be made more precise in that for an ontology $\mathcal{O}$ and entailment η, the set of *reasons* for $\mathcal{O} \models \eta$ is given by the set of justifications $\mathsf{laconic}(\delta^+(\mathcal{O}), \eta)$, where $\mathsf{laconic}$ filters out the justifications that are laconic (see Definition 14 on Page 161). Taking into consideration the various types of masking ensures that all of the reasons for an entailment are captured.

  In essence, each reason should have a “corresponding” laconic justification, that summarises the reasons and makes it possible to determine the original “source” axioms that are responsible for the reason. The notion of a “corresponding” laconic justification will be pinned down later in this thesis, but for now it suffices to consider the following example: Given,

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B \sqcap \exists R.C\\ &\exists R.\top \sqsubseteq B\} \models A \sqsubseteq B\end{aligned}$$

there is one regular justification for $\mathcal{O} \models A \sqsubseteq B$,

$$\mathcal{J} = \{A \sqsubseteq B \sqcap \exists R.C\}$$

However, by Definition 17, external masking is present over $\mathcal{O}$ for $\mathcal{O} \models A \sqsubseteq B$, and this is reflected by the fact that there are intuitively *two reasons* for $\mathcal{O} \models A \sqsubseteq B$—computing justifications over $\delta^{+}(\mathcal{O})$ reveals a set of justifications in which there are two laconic justifications that represent the two reasons, i.e.

$$\begin{aligned}\mathcal{J}_1 = \{&A \sqsubseteq X_1\\ &X_1 \sqsubseteq X_2\\ &X_2 \sqsubseteq X_5\\ &X_5 \sqsubseteq B\}\end{aligned}$$

and

$$\begin{aligned}\mathcal{J}_2 = \{&A \sqsubseteq X_1\\ &X_1 \sqsubseteq X_2\\ &X_2 \sqsubseteq X_4\\ &X_4 \sqsubseteq \exists R.\top\\ &\exists R.\top \sqsubseteq X_6\\ &X_6 \sqsubseteq X_7\\ &X_7 \sqsubseteq B\}\end{aligned}$$

Intuitively, these two “reason representing” justifications *correspond* to

$$\mathcal{J}_1' = \{A \sqsubseteq B\}$$

and

$$\mathcal{J}_2' = \{A \sqsubseteq \exists R.\top, \exists R.\top \sqsubseteq B\}$$

respectively. $\mathcal{J}_1'$ and $\mathcal{J}_2'$ reveal the laconic *sources* of the reasons, and the axioms in $\mathcal{J}_1'$ and $\mathcal{J}_2'$ can be traced back to axioms in $\mathcal{O}$ to reveal the sources of the reasons in $\mathcal{O}$.

- **Repair complete**—It should be possible to repair an ontology from the set of preferred laconic justifications. This desirable feature is tied to the notion of being reason complete. If there is a preferred laconic justification that corresponds to each reason for an entailment, where reason is as defined above, this makes it possible to determine source axioms for each part in a preferred laconic justification and hence makes it possible to devise a repair.

The above criteria, in particular, the notion of reason completeness, suggest the use of δ^+ for obtaining the preferred laconic justifications for an entailment. Indeed δ^+ is rather attractive in this regard: for an ontology $\mathcal{O} \models \eta$, there is a finite subset of $\delta^+(\mathcal{O})$ that contains all the justifications for η hence, the set of justifications with respect to this set is *finite*, δ^+ is based on the structural transformation, which preserves the structure of axioms in $\mathcal{O}$, and finally, δ^+ contains *all* justifications that represent the *reasons* for $\mathcal{O} \models \eta$.

Another attractive feature of δ^+ axioms is that, because δ^+ is a structure preserving transformation, it is possible to *unfold* axioms in a δ^+ justification $\mathcal{J}$ to give a justification $\mathcal{J}'$ which obviously contains axioms that are syntactically close to axioms in $\mathcal{O}$. As an example consider the following ontology

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq \exists R.C \sqcap \forall R.C\\ &B \equiv \exists R.\top\} \models A \sqsubseteq B\end{aligned}$$

Intuitively, a preferred laconic justification for $\mathcal{O} \models A \sqsubseteq B$ is:

$$\mathcal{J} = \{A \sqsubseteq \exists R.\top, \exists R.\top \sqsubseteq B\}$$

where the first axiom is derived from the first axiom in $\mathcal{O}$ and the second axiom is derived from the second axiom in $\mathcal{O}$. Intuitively, $\mathcal{J}$ is a preferred laconic justification, and this tallies with the fact that it corresponds to a reason for

$\mathcal{O} \models \eta$. Applying the δ^+ transformation to $\mathcal{O}$ gives:

$$\begin{aligned}
\delta^+(\mathcal{O}) = \{A &\sqsubseteq X_1\\
X_1 &\sqsubseteq X_3\\
X_3 &\sqsubseteq \exists R.X_2\\
X_3 &\sqsubseteq \exists R.\top\\
X_2 &\sqsubseteq C\\
B &\sqsubseteq X_7\\
X_7 &\sqsubseteq X_8\\
X_8 &\sqsubseteq \exists R.\top\\
\top &\sqsubseteq X_4\\
\exists R.X_4 &\sqsubseteq X_5\\
X_5 &\sqsubseteq X_6\\
X_6 &\sqsubseteq B\}
\end{aligned}$$

From which, two justifications for $\delta^+(\mathcal{O}) \models A \sqsubseteq B$ can be obtained as:

$$\begin{aligned}
\mathcal{J}_1 = \{A &\sqsubseteq X_1\\
X_1 &\sqsubseteq X_3\\
X_3 &\sqsubseteq \exists R.X_2\\
\top &\sqsubseteq X_4\\
\exists R.X_4 &\sqsubseteq X_5\\
X_5 &\sqsubseteq X_6\\
X_6 &\sqsubseteq B\}
\end{aligned}$$

and

$$\begin{aligned}\mathcal{J}_2 = \{A &\sqsubseteq X_1\\ X_1 &\sqsubseteq X_3\\ X_3 &\sqsubseteq \exists R.\top\\ \top &\sqsubseteq X_4\\ \exists R.X_4 &\sqsubseteq X_5\\ X_5 &\sqsubseteq X_6\\ X_6 &\sqsubseteq B\}\end{aligned}$$

Note that $\mathcal{J}_2$ is a weaker form of $\mathcal{J}_1$, as $\mathcal{J}_2$ contains the same axioms as $\mathcal{J}_1$ with the exception of $X_3 \sqsubseteq \exists R.\top$, which a weakening of $X_3 \sqsubseteq \exists R.X_2$. Both $\mathcal{J}_1$ and $\mathcal{J}_2$ can be unfolded to give:

$$\mathcal{J}_1' = \{A \sqsubseteq \exists R.X_2, \exists R.\top \sqsubseteq B\}$$

and

$$\mathcal{J}_2' = \{A \sqsubseteq \exists R.\top, \exists R.\top \sqsubseteq B\}$$

respectively. In this case, $\mathcal{J}_1'$ is *not* laconic, and $\mathcal{J}_2'$ is laconic. Hence, $\mathcal{J}_2'$ corresponds to a preferred laconic justification. Notice that the first axiom in $\mathcal{J}_2'$ is syntactically similar to the first axiom in $\mathcal{O}$ and the second axiom in $\mathcal{J}_2'$ is syntactically similar to the second axiom in $\mathcal{O}$, as desired.

10.3 Computing Preferred Laconic Justifications Using δ^+

Based on the above ideas, it is possible to derive an algorithm for computing preferred laconic justifications that is based the use of δ^+ and the unfolding/replacement technique. For $\mathcal{O} \models \eta$, the basic strategy for computing preferred laconic justifications is to (1) compute the regular justifications for η with respect to $\delta^+(\mathcal{O})$, (2) Process the set of justifications by unfolding fresh concept names that were introduced as part of the δ^+ transformation, and (3) pick out the justifications that are laconic, which correspond to preferred laconic justifications.

10.3.1 An Algorithm for Computing a Finite Subset of δ^+

An algorithm for computing a *finite* subset of $\delta^+(\mathcal{O})$ is presented in Algorithm 10.2. Given a (finite) set of axioms $\mathcal{O}$ as an input, this algorithm first computes $\delta(\mathcal{O})$, which it adds to the result set, and then proceeds to compute weaker forms of axioms in $\delta(\mathcal{O})$. A basic optimisation, whereby axioms are only examined and weakened if they are of the forms that can be weakened to axioms that are non-tautological axioms is employed. For example, $A \sqsubseteq \exists R.B$ can be weakened to the non-tautological axiom $A \sqsubseteq \exists R.\top$, where as $\exists R.B \sqsubseteq A$ cannot be weakened (in a δ-isomorphic compliant fashion) without weakening it to the tautological axiom $\exists R.\bot \sqsubseteq A$, which would never appear in a justification. Algorithm 10.2 requires one sub-routine, GetCardinalityBound which is not defined, but provides an upper bound on the numbers that can appear in cardinality restrictions, and corresponds to the maximum number that appears in cardinality restrictions in an ontology. As explained in Section 9.4.1 this upper bound is provided by $\mathcal{SHOIQ}$'s model theory and ensures a finite subset of $\delta^+(\mathcal{O})$ containing all justifications can be computed.

Lemma 6 (Algorithm 10.2 Termination)**.** *For a finite input $\mathcal{O}$, Algorithm 10.2 terminates and computes all axioms in $\delta^+(\mathcal{O})$ that could appear in a justification for an entailment with respect to $\delta^+(\mathcal{O})$ that holds in $\mathcal{O}$.*

Proof. Given that $\mathcal{O}$ is finite, it follows that $\delta(\mathcal{O})$ is finite and the set of axioms O' (line 1) is finite. The algorithm requires a single pass over the finite set of axioms $\mathcal{O}'$. In lines 3–41 the algorithm generates weaker sets of axioms based on the of $\mathcal{O}'$. For each axiom, in $\mathcal{O}'$ either no axioms are generated, or, a finite set of axioms is generated. The generated sets are either obviously finite, or they are finite since GetCardinalityBound (lines 20 and 28) returns a finite positive integer, which means the loops that increment k are terminate. Hence, the algorithm must terminate. □

10.3.2 An Algorithm for Computing Preferred Laconic Justifications Based on δ^+

Algorithm 10.3 is an algorithm for computing preferred laconic justifications using the δ^+ transformation. It requires four sub-routines: ComputeDeltaPlus, as defined in Algorithm 10.2; ComputeJustifications, which computes regular justifications for $S \models \eta$—any algorithm for computing regular justifications described

Algorithm 10.2 ComputeDeltaPlus($\mathcal{O}$)

```
1: $\mathcal{O}' \leftarrow$ ComputeDelta($\mathcal{O}$)
2: $S \leftarrow \mathcal{O}'$
3: for $\alpha \in \mathcal{O}'$ do
4:     if $\alpha$ is of the form $A \sqsubseteq \exists R.B$ then
5:         $S \leftarrow S \cup \{A \sqsubseteq \exists R.\top\}$
6:     end if
7:     if $\alpha$ is of the form $\forall R.B \sqsubseteq A$ then
8:         $S \leftarrow S \cup \{\forall R.\bot \sqsubseteq A\}$
9:     end if
10:    if $\alpha$ is of the form $A \sqsubseteq \geq nR.B$ then
11:        $k \leftarrow n$
12:        while $k > 0$ do
13:            $S \leftarrow S \cup \{A \sqsubseteq \geq kR.B\}$
14:            $S \leftarrow S \cup \{A \sqsubseteq \geq kR.\top\}$
15:            $k \leftarrow k - 1$
16:        end while
17:    end if
18:    if $\alpha$ is of the form $\geq nR.B \sqsubseteq A$ then
19:        $k \leftarrow n$
20:        $m \leftarrow$ GetCardinalityBound($O$)
21:        while $k \leq m$ do
22:            $S \leftarrow S \cup \{\geq kR.B \sqsubseteq A\}$
23:            $k \leftarrow k + 1$
24:        end while
25:    end if
26:    if $\alpha$ is of the form $A \sqsubseteq \leq nR.B$ then
27:        $k \leftarrow n$
28:        $m \leftarrow$ GetCardinalityBound($O$)
29:        while $k \leq m$ do
30:            $S \leftarrow S \cup \{A \sqsubseteq \geq kR.B\}$
31:            $k \leftarrow k + 1$
32:        end while
33:    end if
34:    if $\alpha$ is of the form $\leq nR.B \sqsubseteq A$ then
35:        $k \leftarrow n$
36:        while $k > 0$ do
37:            $S \leftarrow S \cup \{\leq kR.B \sqsubseteq A\}$
38:            $k \leftarrow k - 1$
39:        end while
40:    end if
41: end for
42: return $S$
```

in Chapter 3 can be used; UnfoldFreshNames, which given a set of axioms S and a signature Σ, unfolds occurrences of names in S which are not in Σ; and, finally, IsLaconic, which was previously defined in Algorithm 10.1.

Lemma 7 (Algorithm 10.3 Termination)**.** *For a finite input $\mathcal{O}$ and η such that $\mathcal{O} \models \eta$, Algorithm 10.3 terminates.*

Proof. The algorithm depends on four sub-routines: ComputeDeltaPlus, which according to Lemma 6, terminates for a finite input $\mathcal{O}$; ComputeJustifications, which is assumed to terminate for a finite input $\mathcal{O}$; IsLaconic, which according to Lemma 4, terminates for an input of $\mathcal{J}$ and η such that $\mathcal{J} \models \eta$; and finally, UnfoldFreshNames, which is not defined, but can safely be assumed to terminate as each fresh name represents an individual occurrence of a sub-concept in a justification, and the definition of a fresh name is therefore acyclic. Beyond the sub-routines, since ComputeJustifications returns a finite set of justifications for $\mathcal{O}' \models \eta$, the loop which filters out laconic justifications (lines 4–9) must therefore terminate. Hence the algorithm terminates. □

Algorithm 10.3 ComputePreferredLaconicJustifications(O, η)

```
Require: $\mathcal{O} \models \eta$
  $O' \leftarrow$ ComputeDeltaPlus($O$)
  $S \leftarrow$ ComputeJustifications($O'$, $\eta$)
  $S' \leftarrow \emptyset$
  for $J \in S$ do
      $J' \leftarrow$ UnfoldFreshNames($J$, signature($O$))
      if IsLaconic($J', \eta$) then
          $S' \leftarrow S' \cup \{J'\}$
      end if
  end for
  return $S'$
```

10.3.3 Cross-Masking and Algorithm 10.3

In the presence of cross-masking, Algorithm 10.3 may suffer from unsatisfactory performance. This is due to an exponential blowup that can occur in the number of *reasons* in the presence of cross-masking. As an illustrative example, consider:

$$\begin{aligned}
\mathcal{O} = \{A &\sqsubseteq B_1 \sqcap \cdots \sqcap B_n \sqcap C_1 \\
&\vdots \\
A &\sqsubseteq B_1 \sqcap \cdots \sqcap B_n \sqcap C_m \\
&\quad B_1 \sqcap \cdots \sqcap B_n \sqsubseteq B\} \models A \sqsubseteq B
\end{aligned}$$

As n grows linearly, the number of justifications for $\delta^+(\mathcal{O}) \models A \sqsubseteq B$ grows as m^n i.e. exponentially. This is in contrast with number of justifications for $\mathcal{O} \models A \sqsubseteq B$, which stays constant with n and grows linearly with m. In terms of computing justifications, this can be problematic, even for small values of n (and m), since, in the worst case, the size of the hitting set tree that must be explored grows exponentially with the number of justifications. Where as it is practical to compute all justifications respect to $\mathcal{O}$ for large values of n and m, the same cannot be said for computing all justification with respect to $\delta^+(\mathcal{O})$.

10.4 An Alternative to δ^+: π

For the purposes of computing preferred laconic justifications it is not necessary to *expose* every *reason* explicitly. Therefore, it is possible to use an optimisation which "bypasses" $\delta^+(\mathcal{O})$, and ultimately prevents the "explosion" of that results from computing justifications with respect to $\delta^+(\mathcal{O})$. The optimisation involves computing justifications with respect to a *representative of the deductive closure* of an ontology. For an ontology $\mathcal{O}$, the representative, $\pi(\mathcal{O})$, contains *weaker* and *shorter* axioms that are syntactically close to axioms in $\mathcal{O}$. The basic idea is that axioms in $\pi(\mathcal{O})$ are generated by the *systematic elimination* and *weakening* of parts of axioms in $\mathcal{O}$. Hence $\pi(\mathcal{O})$ contains axioms that could appear in preferred laconic justifications for some entailment in $\mathcal{O}$. This idea is best illustrated with an example, consider:

$$\begin{aligned}
\mathcal{O} = \{A &\sqsubseteq \exists R.(C \sqcap D) && (\alpha_1) \\
E \sqcup \exists R.C &\sqsubseteq B\} && (\alpha_2)
\end{aligned}$$

The axioms in $\mathcal{O}$ can be weakened in a stepwise manner to give

$$\begin{array}{rll}
\pi(\mathcal{O}) = \{A \sqsubseteq \exists R.(C \sqcap D) & & (\alpha_1) \\
\rightsquigarrow A \sqsubseteq \exists R.C & & (\alpha_{1.1} \text{ from } \alpha_1) \\
\rightsquigarrow A \sqsubseteq \exists R.D & & (\alpha_{1.2} \text{ from } \alpha_1) \\
\rightsquigarrow A \sqsubseteq \exists R.\top & & (\alpha_{1.3} \text{ from } \alpha_{1.1} \text{ or } \alpha_{1.2}) \\
E \sqcup \exists R.C \sqsubseteq B & & (\alpha_2) \\
\rightsquigarrow E \sqsubseteq B & & (\alpha_{2.1} \text{ from } \alpha_2) \\
\rightsquigarrow \exists R.C \sqsubseteq B\} & & (\alpha_{2.2} \text{ from } \alpha_2)
\end{array}$$

where $\alpha_{1.1}$, $\alpha_{1.2}$ and $\alpha_{1.3}$ are weakened forms of $A \sqsubseteq \exists R.(C \sqcap D)$, and $\alpha_{2.1}$ and $\alpha_{2.2}$ are weakened forms of $E \sqcup \exists R.C \sqsubseteq B$. In this example, $\mathcal{O} \models A \sqsubseteq B$, which has a justification with respect to $\mathcal{O}$ of

$$\mathcal{J} = \{A \sqsubseteq \exists R.(C \sqcap D),\ E \sqcup \exists R.C \sqsubseteq B\}$$

has the following justifications with respect to $\pi(\mathcal{O})$:

$$\mathcal{J}_1 = \{A \sqsubseteq \exists R.(C \sqcap D),\ \exists R.C \sqsubseteq B\}$$

and

$$\mathcal{J}_2 = \{A \sqsubseteq \exists R.C,\ \exists R.C \sqsubseteq B\}$$

In this case $\mathcal{J}_1$ is *not* laconic and $\mathcal{J}_2$ *is* laconic for $\mathcal{O} \models A \sqsubseteq B$. Moreover, $\mathcal{J}_2$ corresponds to a preferred laconic justification, as it is the laconic justification that would be obtained by computing justifications with respect to $\delta^+(\mathcal{O})$ followed by unfolding. Indeed, the axioms in $\pi(\mathcal{O})$ are the axioms that would be obtained after unfolding *subsets* of $\delta^+(\mathcal{O})$, and replacing any remaining fresh names with $\top$ or $\bot$ depending on their polarity.

The major benefit of computing justifications with respect to $\pi(\mathcal{O})$ rather than $\delta^+(\mathcal{O})$, is that not all reasons due to cross masking are explicitly exposed and, in the end, the $\pi(\mathcal{O})$ contains the laconic justifications that would get computed using the δ^+ based unfolding and replacement technique described in Section 10.3.2.

10.5 The π Transformation

Definition 19 makes the notion of the π transformation of a set of axioms more concrete, and presents an inductive definition for the function π for $\mathcal{SHOIQ}$ ontologies. Three weakening functions are used: σ, which weakens axioms; τ, which weakens class expressions that have a positive polarity; and β, which weakens class expressions that have a negative polarity. In positive polarity class expressions τ systematically replaces direct subconcepts with $\top$, while in negative polarity class expressions β replaces direct subconcepts with $\bot$.

Definition 19 ($\pi(\mathcal{O})$)**.** *For a set of axioms $\mathcal{O}$ let $\pi(\mathcal{O}) = \mathcal{O} \cup \{\alpha' \mid \alpha' \in \sigma(\alpha) \text{ where } \alpha \in \mathcal{O}\}$. The mappings $\sigma(\alpha)$, $\tau(C)$ and $\beta(C)$ are defined inductively as follows, where n' represents the maximum number in cardinality restrictions in $\mathcal{O}$. The mapping σ weakens axioms by splitting them apart when appropriate and by weakening positive occurrences of sub-concepts with the τ mapping, and negative occurrences of sub-concepts with the β mapping.*

$$\begin{aligned}
\sigma(C \equiv D) &:= \sigma(C \sqsubseteq D) \cup \sigma(D \sqsubseteq C)\\
\sigma(R \equiv S) &:= \{R \sqsubseteq S, S \sqsubseteq R\}\\
\sigma(C(a)) &:= \{C'(a) \mid C' \in \tau(C)\}\\
\sigma(R(a,b)) &:= \sigma(\{a\} \sqsubseteq \exists R.\{b\}) \cup \{R(a,b)\}\\
\sigma(C \sqsubseteq D) &:= \{C' \sqsubseteq D' \mid C' \in \beta(C), D' \in \tau(D)\}\\
\sigma(\alpha) &:= \{\alpha\} \textit{ for any other axiom}
\end{aligned}$$

For positive occurrences of concepts τ weakens each nested sub-concept, and finally weakens the concept itself to $\top$

$$\begin{aligned}
\tau(C_1 \sqcap \cdots \sqcap C_n) &:= \{C'_1 \sqcap \cdots \sqcap C'_n \mid C'_i \in \tau(C_i), 1 \leq i \leq n\}\\
\tau(C_1 \sqcup \cdots \sqcup C_n) &:= \{C'_1 \sqcup \cdots \sqcup C'_n \mid C'_i \in \tau(C_i), 1 \leq i \leq n\}\\
\tau(\neg C) &:= \{\neg C' \mid C' \in \beta(C)\} \cup \{\top\}\\
\tau(\exists R.C) &:= \{\exists R.C' \mid C' \in \tau(C)\} \cup \{\top\}\\
\tau(\forall R.C) &:= \{\forall R.C' \mid C' \in \tau(C)\} \cup \{\top\}\\
\tau(\geq nR.C) &:= \{\geq mR.C' \mid C' \in \tau(C), n \leq m \leq n'\} \cup \{\top\}\\
\tau(\leq nR.C) &:= \{\leq mR.C' \mid C' \in \beta(C), 0 \leq m \leq n\} \cup \{\top\}
\end{aligned}$$

$$\begin{aligned}\tau(A) &:= \{\top, A\}\\ \tau(\{o\}) &:= \{\top, \{o\}\}\end{aligned}$$

For negative occurrences of concepts β weakens each nested sub-concept, and finally weakens the concept itself to $\bot$

$$\begin{aligned}
\beta(C_1 \sqcap \cdots \sqcap C_n) &:= \{C'_1 \sqcap \cdots \sqcap C'_n \mid C'_i \in \beta(C_i), 1 \leq i \leq n\}\\
\beta(C_1 \sqcup \cdots \sqcup C_n) &:= \{C'_1 \sqcup \cdots \sqcup C'_n \mid C'_i \in \beta(C_i), 1 \leq i \leq n\}\\
\beta(\neg C) &:= \{\neg C' \mid C' \in \tau(C)\} \cup \{\bot\}\\
\beta(\exists R.C) &:= \{\exists R.C' \mid C' \in \beta(C)\} \cup \{\bot\}\\
\beta(\forall R.C) &:= \{\forall R.C' \mid C' \in \beta(C)\} \cup \{\bot\}\\
\beta(\geq nR.C) &:= \{\geq mR.C' \mid C' \in \beta(C), n \leq m \leq n'\} \cup \{\bot\}\\
\beta(\leq nR.C) &:= \{\leq mR.C' \mid C' \in \tau(C), 0 \leq m \leq n\} \cup \{\bot\}\\
\beta(\{o\}) &:= \{\bot, \{o\}\}\\
\beta(A) &:= \{\bot, A\}
\end{aligned}$$

In the algorithms that are presented later, the function ComputePi($\mathcal{O}$) computes the axioms that appear in $\pi(\mathcal{O})$ of $\mathcal{O}$ in accordance with Definition 19. For some of the later algorithms it is necessary to trace each axiom that appears in ComputeOPi($\mathcal{O}$) back to the set of axioms in $\mathcal{O}$ from which it was derived. The ComputePi($\mathcal{O}$) function therefore labels each axiom with an index set whose elements point to axioms in $\mathcal{O}$. For example, given

$$\mathcal{O} = \{A \sqsubseteq \exists R.C, A \sqsubseteq \exists R.D\}$$

the ComputePi($\mathcal{O}$) function generates

$$\begin{aligned}\pi(\mathcal{O}) = \{&A \sqsubseteq \exists R.C_{\{1\}}\\ &A \sqsubseteq \exists R.D_{\{2\}}\\ &A \sqsubseteq \exists R.\top_{\{1,2\}}\\ &\quad\ldots\\ &\quad\ldots\quad\}\end{aligned}$$

The dots represent other tautological axioms, for example $A \sqsubseteq \top_{1,2}$, that are not of any particular interest. Notice that the third axiom in $\pi(\mathcal{O})$ can be derived from both the first and second axioms in $\mathcal{O}$. Moreover, in any " weakening step", the labels from one axiom get propagated down to its weaker axioms.

10.6 Choices Regarding the π Transformation

For any ontology $\mathcal{O} \models \eta$, it is important to consider the laconic justifications that are contained in $\pi(\mathcal{O})$. This because it is necessary to decide whether all of these justifications, or some subset of them are *preferred* laconic justifications for $\mathcal{O} \models \eta$. The following examples illustrate how $\pi(\mathcal{O})$ can contain preferred *and* non-preferred laconic justifications for an entailment. Consider

$$\mathcal{O} = \{A \sqsubseteq G \sqcap \exists R.(C \sqcap D),\ \exists R.C \sqcap \exists R.D \sqsubseteq B\} \models A \sqsubseteq B$$

which means that,

$$\begin{aligned}\pi(\mathcal{O}) \supset \{&A \sqsubseteq G \sqcap \exists R.(C \sqcap D)\\ &A \sqsubseteq \exists R.(C \sqcap D)\\ &A \sqsubseteq \exists R.C\\ &A \sqsubseteq \exists R.D\\ &\exists R.C \sqcap \exists R.D \sqsubseteq B\}\end{aligned}$$

There are multiple justifications for $\pi(\mathcal{O}) \models A \sqsubseteq B$, and out of these, there are *two* laconic justifications:

$$\mathcal{J}_1 = \{A \sqsubseteq \exists R.(C \sqcap D),\ \exists R.C \sqcap \exists R.D \sqsubseteq B\}$$

and

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

It is conceivable that either of these justifications could be preferred laconic justifications for $\mathcal{O} \models A \sqsubseteq B$. After all, they both are syntactically related to the axioms in $\mathcal{O}$, and they both reflect the single reason for $\mathcal{O} \models A \sqsubseteq B$.[1] In the

[1]Recall that a reason for $\mathcal{O} \models \eta$ is an occurrence of a justification in $\delta^+(\mathcal{O})$ for η that is laconic

case of $\mathcal{J}_1$ two axioms represent the reason, while in the case of $\mathcal{J}_2$ three axioms represent the reason. However, it is arguable that $\mathcal{J}_1$ and $\mathcal{J}_2$ should not *both* be preferred laconic justifications, since from the usability standpoint, presenting both of them to a person may lead that person to assume there is more than one reason for $\mathcal{O} \models A \sqsubseteq B$, when (in this case) there is not. Therefore, as this example demonstrates, it can be necessary to select a *subset* of justifications that are laconic in $\pi(\mathcal{O})$ as preferred laconic justifications.

10.6.1 Understanding versus Repair

In order to decide whether a subset of $\pi(\mathcal{O})$ laconic justifications is a set of preferred laconic justifications, it is informative to consider how preferred laconic justifications will ultimately be used. In other words, the application in question may determine the outcome. Returning to the question of whether $\mathcal{J}_1$ or $\mathcal{J}_2$ is a preferred laconic justification in the example above, for the purposes of *understanding* (of how parts of axioms in $\mathcal{O}$ entail η) $\mathcal{J}_1$ is probably preferable to $\mathcal{J}_2$. This is because $\mathcal{J}_1$ is structurally closer to $\mathcal{O}$—the nested conjunction is preserved with $\mathcal{J}_1$, while it is absent in $\mathcal{J}_2$. However, for the purposes of *repair*, $\mathcal{J}_2$ is probably preferable to $\mathcal{J}_1$. This is because some of the axioms in $\mathcal{J}_2$ are smaller (and weaker) than the axioms in $\mathcal{J}_1$, and they are therefore conducive to indicating potential repairs in a more direct way that the axioms in $\mathcal{J}_1$. For example, $\mathcal{J}_2$ indicates that either $A \sqsubseteq \exists R.C$ or $A \sqsubseteq \exists R.D$ could be removed, or weakened, in order to break $\mathcal{O} \models A \sqsubseteq B$. It is easy to trace either of these axioms back to axioms in $\mathcal{O}$ in order to figure out how the asserted axioms in $\mathcal{O}$ can be manipulated to enact the repair.

In essence, π justifications which contain axioms that are structurally closer to asserted axioms are preferred for the purposes of understanding, while π justifications that contain weaker and shorter axioms are preferred for the purposes of repair.

10.6.2 Axiom Strength as a Guiding Principle

In the example above, it is noticeable that, while $\mathcal{J}_1$ and $\mathcal{J}_2$ are laconic, some of the axioms in $\mathcal{J}_2$ are weaker than the axioms in $\mathcal{J}_1$, and in fact, $\mathcal{J}_2$ as a whole is weaker than $\mathcal{J}_1$. The fact that $\mathcal{J}_2$ appears in the set of $\pi(\mathcal{O})$ laconic justifications is, in some respect, a side effect of the way $\pi(\mathcal{O})$ is constructed.

Nevertheless, depending on the application and use of laconic justifications, as the above example demonstrates, this side effect can be useful. On the surface, it may appear that given two $\pi(\mathcal{O})$ laconic justifications $\mathcal{J}_a$ and $\mathcal{J}_b$, where $\mathcal{J}_a \models \mathcal{J}_b$ and $\pi(\mathcal{J}_a) \supset \pi(\mathcal{J}_b)$, a general "rule of thumb" is that $\mathcal{J}_a$ should be preferred for understanding and $\mathcal{J}_b$ should be preferred for repair. However, this rule of thumb breaks down when when an ontology contains axioms that entail other axioms. Consider the previous ontology augmented with two axioms that are weaker forms of the first axiom, so that:

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq G \sqcap \exists R.(C \sqcap D)\\ &A \sqsubseteq \exists R.C\\ &A \sqsubseteq \exists R.D\\ &\exists R.C \sqcap \exists R.D \sqsubseteq B\} \models A \sqsubseteq B\end{aligned}$$

In this case, $\pi(\mathcal{O})$ is as before, and $\mathcal{J}_1$ and $\mathcal{J}_2$ from the previous example are still the only two justifications for $\pi(\mathcal{O}) \models \eta$ that are laconic justifications. For the purposes of repair, it is arguable that $\mathcal{J}_2$ alone suffices, since it is easy to trace the axioms in $\mathcal{J}_2$ directly back to the second and third axioms in $\mathcal{O}$ and also indirectly back to the first axiom in $\mathcal{O}$. However, this time, in contrast to the previous example, for the purposes of understanding, both $\mathcal{J}_1$ *and* $\mathcal{J}_2$ can be considered to be preferred laconic justifications, as $\mathcal{J}_1$ derives from the first and fourth axioms in $\mathcal{O}$ while $\mathcal{J}_2$ derives from the second, third and fourth axioms in $\mathcal{O}$. There are two reasons as to why $\mathcal{O} \models A \sqsubseteq B$ and these are reflected in $\mathcal{J}_1$ and $\mathcal{J}_2$. Hence, for the purposes of understanding, $\pi(\mathcal{O})$ laconic justifications which are weaker forms of other $\pi(\mathcal{O})$ laconic justifications cannot simply be marked as non-preferred laconic justifications and discarded.

10.6.3 Choosing Between Understanding and Repair

In summary, when it comes to using laconic justifications for understanding, given a set of $\pi(\mathcal{O})$ laconic justifications for $\mathcal{O} \models \eta$, it is necessary to consider how these justifications relate back to axioms in $\mathcal{O}$ and ultimately consider how these justifications relate to the *reasons* for $\mathcal{O} \models \eta$. When it comes to repair, it suffices to consider the weakest $\pi(\mathcal{O})$ justifications that are laconic.

The remainder of this chapter considers laconic justifications from the point of view of understanding. This choice was made for two reasons: (1) The bulk of this thesis is focused on using justifications as explanations for understanding why entailments hold; and (2) From a computational point of view, selecting laconic justifications for understanding is more challenging than selecting laconic justification for repair. Hence, if in practice it is feasible to compute laconic justifications for understanding, then it ought to be feasible to select laconic justifications for repair.

10.6.4 The Relationship Between π and δ^+

The π transformation parallels the δ^+ transformation in that given an ontology $\mathcal{O}$, for each axiom $\alpha \in \pi(\mathcal{O})$ there is some subset of $\delta^+(\mathcal{O})$ that can be unfolded and manipulated into α. This is fairly easy to see from the way the rewrite rules in Definition 13 and the transformation function in Definition 19 are specified. Hence, $\pi(\mathcal{O})$ contains all "reason representing" justifications for any entailment η in $\mathcal{O}$ and can be used for computing preferred laconic justifications for understanding.

10.7 Computing Preferred Laconic Justifications Using π

Algorithm 10.4 is an algorithm for computing preferred laconic justifications based on π. The algorithm requires four sub-routines: (1) ComputePi, which, as described above, for an input $\mathcal{O}$ generates a set of labelled axioms contained in $\pi(\mathcal{O})$; (2) ComputeJustifications, which given $\mathcal{O}$ and η computes the regular justifications for $\mathcal{O} \models \eta$; (3) isLaconic which determines whether a justification is a laconic justification for a given entailment, and is described in Algorithm 10.1; and finally, (4) isPreferredLaconic, which determines if a justification $\mathcal{J}$ for an entailment η is a preferred laconic justification with respect to an ontology $\mathcal{O}$.

The algorithm proceeds as follows: Given an ontology $\mathcal{O}$ and an entailment η such that $\mathcal{O} \models \eta$, the algorithm first uses the sub-routine ComputePi to compute $\pi(\mathcal{O})$ for $\mathcal{O}$. Next, it computes justifications for η with respect to $\pi(\mathcal{O})$, and out of these filters out the justifications that are laconic (lines 4–8), placing them in the set L. The algorithm then uses the isPreferredLaconicJustification sub-routine to filter out the set of preferred laconic justifications from L, which it returns as set R (lines 10–15).

Lemma 8 (Algorithm 10.4 Termination)**.** *For an input ontology $\mathcal{O}$ and entailment η such that $\mathcal{O} \models \eta$ Algorithm 10.4 terminates.*

Proof. Assuming that all sub-routines terminate for their respective inputs, which are all finite, it is easy to see that Algorithm 10.4 terminates, as it contains two two loops (lines 4–8 and 10–15) which iterate over finite sets. The sets are finite because $\pi(O)$ is finite (as $\mathcal{O}$ is finite), and the ComputeJustifications sub-routine therefore returns a finite set of justifications which it places into the set S. □

Algorithm 10.4 ComputePreferredLaconicJustifications($\mathcal{O}$, η)

```
Require: $\mathcal{O} \models \eta$
 1: $\mathcal{O}_\pi \leftarrow$ ComputePi($\mathcal{O}$)
 2: $S \leftarrow$ ComputeJustifications($\mathcal{O}_\pi$, $\eta$)
 3: $L \leftarrow \emptyset$
 4: for $J \in S$ do
 5:     if isLaconic($J$, $\eta$) then
 6:         $L \leftarrow L \cup \{J\}$
 7:     end if
 8: end for
 9: $R \leftarrow \emptyset$
10: for $J \in L$ do
11:     if isPreferredLaconicJustification($J$, $\eta$, $\mathcal{O}$) then
12:         $R \leftarrow R \cup \{J\}$
13:     end if
14: end for
15: return $R$
```

The isPreferredLaconicJustification sub-routine in Algorithm 10.4 is pluggable. Different implementations could be used to compute different sets of preferred laconic justifications for different purposes (understanding or repair for example). Inline with the above discussion on preferred laconic justifications, Algorithm 10.5 can be used to determine if a justification is preferred for the purposes of *understanding* how the parts of axioms in an ontology support an entailment. In essence, the algorithm determines whether or not a laconic justification for $\mathcal{O} \models \eta$ corresponds with a reason, i.e. a justification for $\delta^+(\mathcal{O}) \models \eta$. If this is the case then the algorithm returns true, otherwise, it returns false. Algorithm 10.5 proceeds in the obvious way, by first computing δ^+ of the input ontology $\mathcal{O}$. It then makes calls to a sub-routine GetNextJustification, which is assumed to *incrementally* compute justifications for an entailment in a set of axioms (S_δ

and η in this case). For each justification that is found, it is unfolded using the UnfoldFreshNames sub-routine. If this unfolded justification, J'' is equal to the input justification J then J corresponds to a reason for $\mathcal{O} \models \eta$ and the algorithm returns true i.e. J is a preferred laconic justification. If, after examining all of the δ^+ justifications, J is not a preferred laconic justification, then the algorithm returns false.

Algorithm 10.5 isPreferredLaconicJustification(J, η, $\mathcal{O}$)

```
Require: $J$, $\eta$ and $\mathcal{O}$ such that $\mathcal{O} \models \eta$ and $J$ is a laconic justification $\eta$ over $\mathcal{O}$.
 1: $O' \leftarrow$ GetSourceAxioms($J$)
 2: $S_\delta \leftarrow$ ComputeDeltaPlus($O'$)
 3: repeat
 4:     $J' \leftarrow$ GetNextJustification($S_\delta$, $\eta$)
 5:     $J'' \leftarrow$ UnfoldFreshNames($J'$, signature($\mathcal{O}$))
 6:     if $J'' = J$ then
 7:         return true
 8:     end if
 9: until $J' = \emptyset$
10: return false
```

10.7.1 An Optimised isPreferredLaconicJustification Algorithm

As can be seen from Algorithm 10.5, for *every* input J, η and $\mathcal{O}$, it uses the strategy of *incrementally* computing justifications for $\delta^+(\mathcal{O}) \models \eta$. The algorithm terminates early when it finds a justification in $\delta^+(\mathcal{O})$ which verifies that J is a preferred laconic justification. Therefore, in the case of preferred laconic justifications, it might not be necessary to compute all justifications for $\delta^+(\mathcal{O}) \models \eta$. However, this δ^+ based test is still an expensive test. Fortunately, there is a much cheaper test that can be used to avoid this expensive test in many cases. The cheaper test requires access to all laconic justifications in $\pi(\mathcal{O})$, and so the modification to Algorithm 10.5 requires an extra parameter to carry this information. Algorithm 10.6 is an adaptation of Algorithm 10.5 that includes this extra parameter along with the cheap test, which is contained in lines 2–12. For an input of J and L, the cheap test works by trying to find a laconic justification in L that *might* rule out J from being a preferred laconic justification. It does this by checking whether J contains axioms that are weaker $\pi(\mathcal{O})$ derivatives of some other laconic justification J' (lines 4–6). If this is the case, it is necessary to perform the more expensive test on J and the algorithm continues as before.

Algorithm 10.6 isPreferredLaconicJustification(J, η, $\mathcal{O}$, L)

```
Require: $J$, $\eta$, $\mathcal{O}$ and $L$ where $\mathcal{O} \models \eta$, $J$ is a laconic justification $\eta$ over $\mathcal{O}$, $L$ is
    a set of laconic justifications for $\eta$ over $\mathcal{O}$ where $J \in L$.
 1: $pref \leftarrow$ true
 2: for $J' \in L$ do
 3:     if $J' \neq J$ then
 4:         $J'^{+} \leftarrow$ ComputePi($J'$)
 5:         $J^{+} \leftarrow$ ComputePi($J$)
 6:         if $J^{+} \subset J'^{+}$ then
 7:             $pref \leftarrow$ false
 8:         end if
 9:     end if
10: end for
11: if $pref =$ true then
12:     return true
13: else
14:     $O' \leftarrow$ GetSourceAxioms($J$)
15:     $S_\delta \leftarrow$ ComputeDeltaPlus($O'$)
16:     repeat
17:         $J' \leftarrow$ GetNextJustification($S_\delta$, $\eta$)
18:         $J'' \leftarrow$ UnfoldFreshNames($J'$, signature($\mathcal{O}$))
19:         if $J'' = J$ then
20:             return true
21:         end if
22:     until $J' = \emptyset$
23:     return false
24: end if
```

If the algorithm fails to find such a J', then J must be a preferred laconic justification and the algorithm terminates early (line 12) returning true, and avoiding the more expensive test.

10.8 Optimising π Laconic Justification Computation

Algorithm 10.4 may be seen as an improvement over Algorithm 10.3. In particular, it avoids the exponential blowup that is induced by cross-masking. However, some optimisations to this algorithm are still necessary. This section details several optimisations that are related to the computation of $\pi(\mathcal{O})$, which are contained in Algorithm 10.7.

10.8.1 Top Level Axiom Splitting and Reconstitution

One of the main problems with the unoptimised algorithm and use of $\pi(\mathcal{O})$ is that there can be a far larger number of justifications for $\pi(\mathcal{O}) \models \eta$ in comparison to $\mathcal{O} \models \eta$. The following example illustrates the reason as to why there can be an increase in justifications. Consider,

$$\mathcal{O} = \{A \sqsubseteq B \sqcap C_1 \sqcap \cdots \sqcap C_n\} \models A \sqsubseteq B$$

where it is assumed that the C_i $(1 \leq i \leq n)$ conjuncts play no part in entailing $A \sqsubseteq B$. In this case, $\pi(\mathcal{O})$ contains in the order of 2^n axioms, as each possible subset of conjuncts (and their weaker variants) is taken into consideration. While there is one justification for $A \sqsubseteq B$ in $\mathcal{O}$ there are 2^n justifications for $A \sqsubseteq B$ in $\pi(\mathcal{O})$. For small values of n, which is typically the case for real ontologies, the exponential aspect does not directly represent a problem regarding the size of $\pi(\mathcal{O})$. However, there can be a signification increase in the number of justifications for $A \sqsubseteq B$ in $\pi(\mathcal{O})$ when compared to $\mathcal{O}$, even for small values of n. Indeed, consider the slightly more concrete and realistic axiom patterns in

$$\mathcal{O} = \{A \sqsubseteq \exists R.C \sqcap \exists R.D,\ \ \exists R.\top \sqsubseteq B\} \models A \sqsubseteq B$$

which essentially represents a class expression A consisting of a set of existential restrictions, and a domain axiom which states that the the domain of R is B. There is just one justification for $A \sqsubseteq B$ with respect to $\mathcal{O}$. However, $\pi(\mathcal{O})$ contains

$$\begin{gathered}
A \sqsubseteq \exists R.C \sqcap \exists R.D \\
A \sqsubseteq \exists R.C \sqcap \exists R.\top \\
A \sqsubseteq \exists R.\top \sqcap \exists R.D \\
A \sqsubseteq \exists R.\top \\
A \sqsubseteq \exists R.C \\
A \sqsubseteq \exists R.D \\
\exists R.\top \sqsubseteq B
\end{gathered}$$

which contains *six* justifications for $A \sqsubseteq B$, just one of which is a preferred laconic justification according to the previously developed arguments. Given the exponential nature of the hitting set tree based algorithms for computing all justifications, the large increase in the number of justifications can have obviously have a negative effect. At best, the computation of justifications with respect to $\pi(\mathcal{O})$ takes (much) longer than it does when computing justifications with respect to $\mathcal{O}$, and at worst, it can be *impractical* to compute all $\pi(\mathcal{O})$ justifications in situations where it is practical to compute all justifications with respect to $\mathcal{O}$.

In essence a blow up in the number of axioms in $\pi(\mathcal{O})$ can arise due to the combinatorial explosion of subconcept weakening. An effective optimisation that ameliorates this problem is the top level splitting of subclass axioms. Given an axiom of the form

$$C_1 \sqcup \cdots \sqcup C_n \sqsubseteq D_1 \sqcap \cdots \sqcap D_m$$

it is possible to transform it into an equivalent set of $n \times m$ subclass axioms

$$\{C_i \sqsubseteq D_j \mid 1 \leq i \leq n, 1 \leq j \leq m\}$$

This can dramatically reduce the number of axioms in $\pi(\mathcal{O})$. For example, in the case where C_i and D_j $(1 \leq i \leq n, 1 \leq j \leq m)$ are class names, $\pi(\mathcal{O})$ is of the order of 2^{n+m} axioms in size, where as $\pi(\mathcal{O}_{split})$ is of the order of $n \times m$ axioms in size (disregarding tautological axioms). When C_i and C_j are complex class expressions, the size of $\pi(\mathcal{O}_{split})$ depends upon the degree and complexity of nesting of the class expressions, however, for realistic ontologies, where class expression nesting is not particularly deep and complex, the difference between $|\pi(\mathcal{O})|$ and $|\pi(\mathcal{O}_{split})|$ is obviously significant for moderately large n and m.

This splitting technique can be used in combination with the labelling of axioms in $\pi(\mathcal{O})$ as an optimisation for computing laconic justifications. The following example illustrates how the technique works. Consider the ontology,

$$\mathcal{O} = \{A \sqsubseteq E \sqcap \exists R.C_{\{1\}}, E \sqcap \exists R.\top \sqsubseteq B_{\{2\}}\} \models A \sqsubseteq B$$

where the axioms have been labelled with indices. Computing $\pi(O)$ of $\mathcal{O}$ gives

(disregarding tautological axioms)

$$\begin{aligned}\pi(\mathcal{O}) = \{&A \sqsubseteq E \sqcap \exists R.C_{\{1\}}\\ &A \sqsubseteq \exists R.C_{\{1\}}\\ &A \sqsubseteq \exists R.\top_{\{1\}}\\ &A \sqsubseteq E \sqcap \exists R.\top_{\{1\}}\\ &A \sqsubseteq E_{\{1\}}\\ &E \sqcap \exists R.\top \sqsubseteq B_{\{2\}}\}\end{aligned}$$

which contains *four* justifications for $A \sqsubseteq B$ (two of which are laconic, and one of which is preferred laconic) in comparison to *one* justification in $\mathcal{O}$ for the same entailment. Now, splitting $\mathcal{O}$, and propagating indices gives

$$\mathcal{O}_{split} = \{A \sqsubseteq E_{\{1\}}, A \sqsubseteq \exists R.C_{\{1\}}, A \sqsubseteq \exists R.\top_{\{1\}}, \exists R.\top \sqsubseteq B_{\{2\}}\} \models A \sqsubseteq B$$

and computing $\pi(\mathcal{O}_{split})$ gives (disregarding tautological axioms)

$$\begin{aligned}\mathcal{O}^{+}_{split} = \{&A \sqsubseteq E_{\{1\}}\\ &A \sqsubseteq \exists R.C_{\{1\}}\\ &A \sqsubseteq \exists R.\top_{\{1\}}\\ &E \sqcap \exists R.\top \sqsubseteq B_{\{2\}}\}\end{aligned}$$

which contains two justifications, one of which is laconic:

$$\mathcal{J} = \{A \sqsubseteq E_{\{1\}},\quad A \sqsubseteq \exists R.\top_{\{1\}},\quad E \sqcap \exists R.\top \sqsubseteq B_{\{2\}}\}$$

The index sets can now be used to merge subclass axioms that have *exactly* the same sources. In this case, the axioms that have an index set of $\{1\}$, which gives

$$\mathcal{J}' = \{A \sqsubseteq E \sqcap \exists R.\top_{\{1\}},\quad E \sqcap \exists R.\top \sqsubseteq B_{\{2\}}\}$$

which corresponds to a preferred laconic justification.

As can be seen, given $\mathcal{O} \models \eta$ there are two major advantages to using and computing justifications with respect to $\pi(\mathcal{O}_{split})$ instead of $\pi(\mathcal{O})$: (1) The size of $\pi(\mathcal{O}_{split})$ can be less than the size of $\pi(\mathcal{O})$. In fact, the size of $\pi(\mathcal{O}_{split})$ can be exponentially smaller than the size of $\pi(\mathcal{O})$; (2) The number of justifications for

$\pi(\mathcal{O}_{split}) \models \eta$ can be exponentially smaller than the number of justifications for $\pi(\mathcal{O}) \models \eta$. Even in cases where the exponential aspect of the size of $\pi(\mathcal{O})$ does not dominate, the reduction in the number of justifications that can be achieved when using $\pi(\mathcal{O}_{split})$ can have a significant effect when it comes to computing all justifications.

When dealing with axioms that have multiple sources, it is necessary to take more care so as not to inadvertently discard preferred laconic justifications. Consider the ontology

$$\mathcal{O} = \{A \sqsubseteq B \sqcap C_{\{1\}}, A \sqsubseteq B_{\{2\}}, A \sqsubseteq C_{\{3\}}\} \models A \sqsubseteq B \sqcap C$$

Splitting $\mathcal{O}$ gives,

$$\mathcal{O}_{split} = \{A \sqsubseteq B_{\{1,2\}}, A \sqsubseteq C_{\{1,3\}}\}$$

as axiom 1 gets split into axioms that are the same as axiom 2 and axiom 3. Which means that, disregarding tautological axioms,

$$\mathcal{O}^{+}_{split} = \{A \sqsubseteq B_{\{1,2\}}, A \sqsubseteq C_{\{1,3\}}\}$$

which gives one justification that is laconic, namely $\mathcal{J} = \{A \sqsubseteq B_{\{1,2\}}, A \sqsubseteq C_{\{1,3\}}\}$. Now, although the indices that label axioms in $\mathcal{J}$ indicate there are two subclass axioms that share the same source axiom, it does not suffice to simply merge these two axioms to yield $\mathcal{J}' = \{A \sqsubseteq B \sqcap C\}$. Doing this would cause a preferred laconic justification, $\mathcal{J}'' = \{A \sqsubseteq B,\ A \sqsubseteq C\}$ to be discarded. This situation arises when in a justification two or more axioms share the same source axiom, and so can be reconstituted, but also share other source axioms. To deal with this, the reconstitution process must generate a set of justifications from the original axioms with the reconstituted axioms. In the above example, the set $S = \{A \sqsubseteq B \sqcap C_{\{1\}}, A \sqsubseteq B_{\{1,2\}}, A \sqsubseteq C_{\{1,3\}}\}$ is generated, with the axioms that have multiple sources being left intact. Next justifications for $A \sqsubseteq B$ are computed with respect to S. This gives $\mathcal{J}_1 = \{A \sqsubseteq B_{\{1,2\}}, A \sqsubseteq C_{\{1,3\}}\}$ and $\mathcal{J}_2 = \{A \sqsubseteq B \sqcap C_{\{1,2,3\}}\}$. The justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ can then be examined to see if $\mathcal{J}_1$ really is a preferred laconic justification.

10.8.2 Pruning via Modularity

As was seen in Chapter 3, the use of modularity techniques, specifically syntactic locality based modules, can provide a significant optimisation when it comes to computing justifications. For an ontology $\mathcal{O}$, where $\mathcal{O} \models \eta$, and the size of a module $\mathcal{M} \subseteq \mathcal{O}$ is much smaller than $\mathcal{O}$, modularisation can obviously be used to obtain a reduction in the size of $\pi(\mathcal{O})$.

10.8.3 Tautological Axiom Elimination

For simplicity, Definition 19 was given inductively on the structure of $\mathcal{SHOIQ}$ axioms and concepts. The result of this is that for a given ontology $\mathcal{O}$, $\pi(\mathcal{O})$ may contain many syntactically detectable tautological axioms that would never appear in any justification for any entailment. For example, given $A \sqsubseteq \exists R.C \sqcap \forall R.C$, amongst other axioms $\pi(\mathcal{O})$ contains $A \sqsubseteq \forall R.\top$, $\bot \sqsubseteq \exists R.C \sqcap \forall R.C$, and $A \sqsubseteq \top$. On the surface, these tautological axioms, may appear harmless. However, large numbers of such axioms can have an impact on black-box justifications finding. There are two main reasons: (1) The axioms bulk up the input for entailment testing. Although reasoners usually employ a suite of syntactic inspection optimisations to detect axioms that are obviously tautological and thus avoid performing expensive tableau tests in many cases, a large number of tautological $\pi(\mathcal{O})$ axioms can increase loading and preprocessing times. Since black-box justification finding can involve significant numbers of entailment tests, the cumulative effect can be significant; (2) Tautological $\pi(\mathcal{O})$ axioms unnecessarily increase the search space in black-box algorithms for computing single justifications. Since the tautological axioms in $\pi(\mathcal{O})$ are syntactically related to each other, and share the same signature with non-tautological axioms, they can increase the number of steps in the expansion phases and contraction phases of black-box justification finding algorithms. This obviously increases the number of entailment tests that are required.

Finally, unnecessary generation of tautological $\pi(\mathcal{O})$ axioms can be very expensive. Consider the axiom $A \sqsubseteq B_1 \sqcup \cdots \sqcup B_n$, where B_i $(1 \leq i \leq n)$ is a class name. According to Definition 19, $\pi(\mathcal{O})$ contains one axiom for each element of the power set of $\{B_1, \ldots, B_n\}$. However, each B_i is a class name, and can only be weakened to $\top$, which means that 2^n tautological axioms wastefully are produced. Therefore, an optimisation is not to compute weakened axioms for axioms of the form $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B_1 \sqcup \cdots \sqcup B_m$ for $1 \leq n, m$

In summary, being cautious about computing and eliminating tautological axioms when computing $\pi(\mathcal{O})$ can have worthwhile positive effects.

10.8.4 An Optimised ComputePreferredLaconicJustifications Algorithm

Algorithm 10.7 is an optimised version of Algorithm 10.4 for computing preferred laconic justifications. It contains each of the optimisations discussed above and includes the optimised isPreferredLaconicJustification algorithm that was presented in Algorithm 10.6. Aside from this, the algorithm requires three sub-routines that were not used in Algorithm 10.4: (1) ComputeModule, which computes a syntactic locality based module for a set of axioms and a signature; (2) ComputeSplitPi, which takes a set of axioms as an input, labels them, splits subclass axioms, propagating the labels to the split axioms, as described above, and then computes $\pi(\mathcal{O})$ from these split axioms; (3) ReconstituteSplitAxioms, which takes a set of labelled, split $\pi(\mathcal{O})$ axioms as an input and reconstitutes them as described above.

Algorithm 10.7 ComputePreferredLaconicJustifications($\mathcal{O}$, η)

```
Require: $\mathcal{O} \models \eta$
 1: $\Sigma_\eta \leftarrow$ signature($\eta$)
 2: $\mathcal{M} \leftarrow$ ComputeModule($\mathcal{O}$, $\Sigma_\eta$)
 3: $\mathcal{O}^+ \leftarrow$ ComputeSplitPi($\mathcal{M}$)
 4: $S_J \leftarrow$ ComputeJustifications($\mathcal{O}^+$, $\eta$)
 5: $L \leftarrow \emptyset$
 6: for $J \in S_J$ do
 7:     if isLaconic($J$, $\eta$) then
 8:         $S_R \leftarrow$ ReconstituteSplitAxioms($J$)
 9:         $L \leftarrow L \cup S_R$
10:     end if
11: end for
12: $R \leftarrow \emptyset$
13: for $J \in L$ do
14:     if isPreferredLaconicJustification($J$, $\eta$, $\mathcal{O}$, $L$) then
15:         $R \leftarrow R \cup \{J\}$
16:     end if
17: end for
18: return $R$
```

10.8.5 An Incremental ComputePreferredLaconicJustifications Algorithm

It will later be seen that Algorithm 10.7 performs well for entailments in many of the BioPortal ontologies. However, there are some ontologies where computing $\pi(\mathcal{O})$ of a module for an entailment signature can be expensive. Algorithm 10.8 offers an incremental approach to computing $\pi(\mathcal{O})$ and finding laconic justifications.

Given an ontology $\mathcal{O}$ and an entailment η where $\mathcal{O} \models \eta$, the basic strategy is to loop, repeatedly computing justifications for η with respect $\mathcal{O} \cup \pi(\mathcal{S})$, where $\mathcal{S}$ is the (initially empty) set of justifications found in the previous loop. The loop continues until no new justifications are found. In each step after the first round of computing justifications, if new justifications are found, they may be externally masked justifications.

The algorithm is best illustrated with an example. Consider the ontology

$$\begin{aligned}\mathcal{O} = \{&A \sqsubseteq B \sqcap \neg B \sqcap C,\\ &A \sqsubseteq \neg C,\\ &A \sqsubseteq D\} \models A \sqsubseteq \bot\end{aligned}$$

Notice that external masking occurs in $\mathcal{O}$ for the entailment $A \sqsubseteq \bot$, and that there is just one regular justifications for this entailment (which is the singleton set containing the first axiom). In the first loop the algorithm computes the set of justifications

$$\mathcal{S}_1 = \{\{A \sqsubseteq B \sqcap \neg B \sqcap C\}\}$$

In the second loop justifications are computed with respect to $\mathcal{O} \cup \pi(\mathcal{S}_1)$, namely[2]

$$\begin{aligned}\mathcal{O} \cup \pi(\mathcal{S}_1) = \{&A \sqsubseteq B \sqcap \neg B \sqcap C,\\ &A \sqsubseteq B\\ &A \sqsubseteq \neg B\\ &A \sqsubseteq C\\ &A \sqsubseteq \neg C,\\ &A \sqsubseteq D\} \models A \sqsubseteq \bot\end{aligned}$$

[2]Note that top level splitting has been used above to make the example simpler.

This gives a new set of justifications

$$\begin{aligned}\mathcal{S}_2 = \{&\{A \sqsubseteq B \sqcap \neg B \sqcap C\}\\ &\{A \sqsubseteq B, A \sqsubseteq \neg B\}\\ &\{A \sqsubseteq C, A \sqsubseteq \neg C\}\}\end{aligned}$$

Since $S_1 \neq \mathcal{S}_2$, the algorithm continues looping, and computes the set of justifications $\mathcal{S}_3$ for η with respect to $\mathcal{O} \cup \pi(\mathcal{S}_2)$. In this case, it finds that $\mathcal{S}_3 = \mathcal{S}_2$ and therefore stops looping. It extracts the set of preferred laconic justifications from $\mathcal{S}_3$, which are

$$\mathcal{J}_1 = \{A \sqsubseteq B \sqcap \neg B \sqcap C\}$$

and

$$\mathcal{J}_2 = \{A \sqsubseteq C, A \sqsubseteq \neg C\}$$

returns these and terminates.

10.9 Conclusions

This chapter has presented a decision procedure for deciding whether a justification is laconic or not, and several algorithms and accompanying optimisations for computing preferred laconic justifications. While the algorithm for the decision procedure more or less falls out of the definition of laconic and precise justifications, the same cannot be said about an algorithm for computing laconic justifications. First, it was necessary to introduce the notion of preferred laconic justifications, which are based on capturing all of the reasons as to why an entailment holds, and then it was necessary to design an optimised algorithm which necessitated bypassing the obvious algorithmic design of computing justifications with respect to axioms resulting from applying the structural transformation. In essence, preferred laconic justification get computed with respect to a filter (π) on the deductive closure of an ontology, which contains axioms that are stepwise weakenings of the source axioms. The filter and parallels the structural transformation and contains all reason representing justifications.

Algorithm 10.8 ComputePreferredLaconicJustifications($\mathcal{O}$, η)

```
Require: $\mathcal{O} \models \eta$
 1: $\Sigma_\eta \leftarrow$ signature($\eta$)
 2: $\mathcal{M} \leftarrow$ ComputeModule($\mathcal{O}$, $\Sigma_\eta$)
 3: $S \leftarrow \emptyset$
 4: $S' \leftarrow \emptyset$
 5: repeat
 6:     $S = S'$
 7:     $W \leftarrow \mathcal{O} \cup$ ComputeSplitOPlus($S$)
 8:     $S' \leftarrow$ ComputeJustifications($W$,$\eta$)
 9: until $S' = S$
10: $L \leftarrow \emptyset$
11: for $J \in S_J$ do
12:     if isLaconic($J$, $\eta$) then
13:         $S_R \leftarrow$ ReconstituteSplitAxioms($J$)
14:         $L \leftarrow L \cup S_R$
15:     end if
16: end for
17: $R \leftarrow \emptyset$
18: for $J \in L$ do
19:     if isPreferredLaconicJustification($J$, $\eta$, $\mathcal{O}$, $L$) then
20:         $R \leftarrow R \cup \{J\}$
21:     end if
22: end for
23: return $R$
```

Chapter 11

Laconic Justification Finding Experiments

This chapter presents a series of experiments that aim to investigate the practicality of detecting and computing preferred laconic justifications using the previously presented algorithms. Because Algorithm 10.1 (IsLaconic) is used as a subroutine in other algorithms it is first evaluated in isolation. Next, two algorithms for computing preferred laconic justification are evaluated and compared: Algorithm 10.7, which is the optimised π based algorithm that uses top level splitting, and Algorithm 10.8, which is the optimised *incremental* π based algorithm that uses top level splitting. In all of the experiments the BioPortal corpus that was described in Chapter 5 was used for testing.

11.1 Detecting Laconic Justifications

Experiment 4 presents an evaluation of Algorithm 10.1 which detects whether or not a justification is laconic. In addition to gaining information about how the algorithm performs on real justifications in real world ontologies, namely the BioPortal ontologies, this experiment also enabled data about the prevalence of laconic and non-laconic regular justifications "in the wild" to be collected.

Experiment 4: Algorithm 10.1, IsLaconic, Empirical Evaluation

Experiment 4 Algorithm Implementation

Algorithm 10.1, including the sub-routine ComputeDelta, was implemented in Java. The sub-routine was implemented according to the rewrite rules presented in Definition 7. The implementation of the algorithm deviated slightly from that presented in listing 10.1. Specifically, the early termination aspect of the algorithm was not implemented. That is, the actual implementation of algorithm examines *every* axiom in S_δ without terminating and returning false as soon as it finds an axiom that is superfluous, or that could be weakened. The reason for this was to enable descriptive statistics to be collected about the occurrences of superfluity in naturally occurring justifications in the BioPortal corpus.

Experiment 4 Test Data

Justifications collected from the output of Experiment 1 were collected and used as an input to the implementation of Algorithm 10.1. This provided a total of 468,819 justifications for 93,975 entailments in 72 ontologies. In addition to this, a pilot experiment indicated that a high number of BioPortal justifications were non-laconic because they contained one or more axioms of the form $C \equiv D$ where only one direction of the implication (i.e. $C \sqsubseteq D$ *or* $D \sqsubseteq C$ but not $C \equiv D$) was required. In order to gain a picture of the prevalence of superfluous sub-concepts alone, each input justification $\mathcal{J}$ that contained one or more equivalent concept axioms was split to yield a set of axioms, and then another justification $\mathcal{J}'$ contained in the split set of axioms was computed and tested. For example $\mathcal{J} = \{A \sqsubseteq B, C \equiv B\} \models A \sqsubseteq B$ was split to give the set of axioms $\mathcal{S} = \{A \sqsubseteq B, B \sqsubseteq C, C \sqsubseteq B\}$, from which $\mathcal{J}' = \{A \sqsubseteq B, B \sqsubseteq C\}$ is obtained. These justifications that were the result of splitting were also used as an input to the implementation of Algorithm 10.1, and provided an additional 389,038 justifications for 93,975 entailments in the set of BioPortal ontologies, thus providing 857,857 real justifications in total.

Experiment 4 Method

The test data were used as input to Algorithm 10.1. For each justification, $\mathcal{J}$, the CPU time required for the implementation of Algorithm 10.1 to return true (laconic) or false (not laconic) was recorded. The number of laconic versus the number of non-laconic justifications was recorded, and, for each justification, the number of axioms containing superfluous parts was also recorded. The experiment run was conducted using the Pellet 2.2.2 reasoner, which was accessed via the OWL API. The experiments were performed on MacBook Pro with a 3.06 GHz Intel Core 2 Duo Processor. The Java Virtual Machine was allocated a maximum of 4 GB of RAM.

Experiment 4 Results

Figure 11.1 shows the percentile times for checking whether a justification is laconic or not, along with mean time per justification. Mean times are shown as white outlines. Across all ontologies, the mean time per justification was 328 ms with a standard deviation = 339 ms. Ontologies are ordered along the x-axis by the 99th percentile time.

Figure 11.2 shows the percentage of non-laconic justifications per ontology. The ontologies are ordered along the x-axis by the percentage of non-laconic split justifications. The light coloured bars show the non-split justifications (as found in the ontology) and the dark coloured bars show percentages of split justifications. Out of the 468,819 justifications 387,713 (82.7%) were *not* laconic. Out of the 389,038 split justifications 239,290 (61.5%) were *not* laconic. There were just three ontologies (Ontologies 31, 44 and 59) all of whose justifications were laconic.

Finally, Figure 11.3 shows the mean number of axioms per non-split justification that are non-laconic i.e. the mean number of axioms that contain superfluous parts. Figure 11.4 shows the mean number of axioms per split justification that are non-laconic. In both figures the white coloured/empty bars represent the mean over all justifications in the associated ontology. In total, for the non-split axioms, there were 36 ontologies where the average number of non-laconic axioms per justification was greater than 1. For the split justifications, there were 17 ontologies where the average number of non-laconic axioms per justification was greater than 1.

Experiment 4 Analysis

Algorithm Performance Figure 11.1 indicates that the implementation of Algorithm 10.1 gives acceptable runtime performance—average case performance

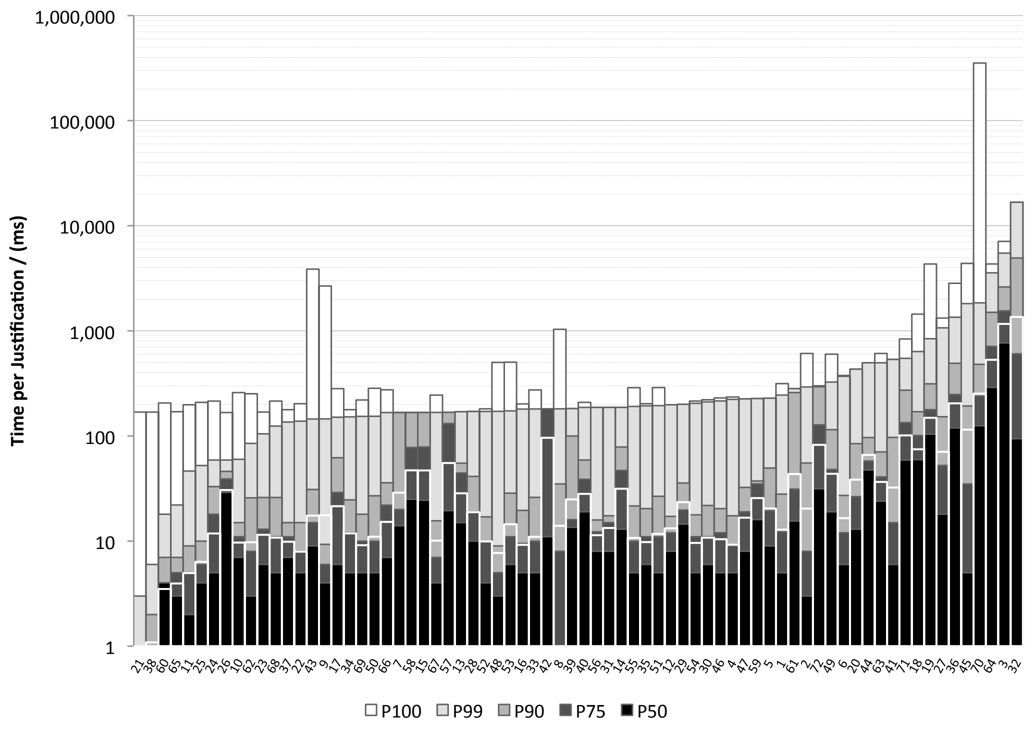


Figure 11.1: Experiment 4—IsLaconic Percentile Times Per Justification

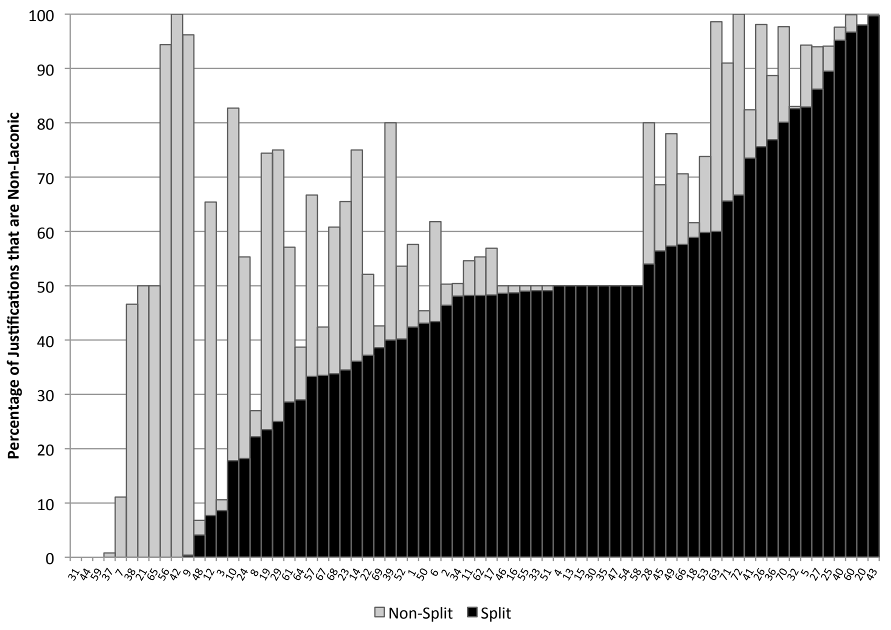


Figure 11.2: Experiment 4—The Percentage of Non-Laconic Justifications Per Ontology

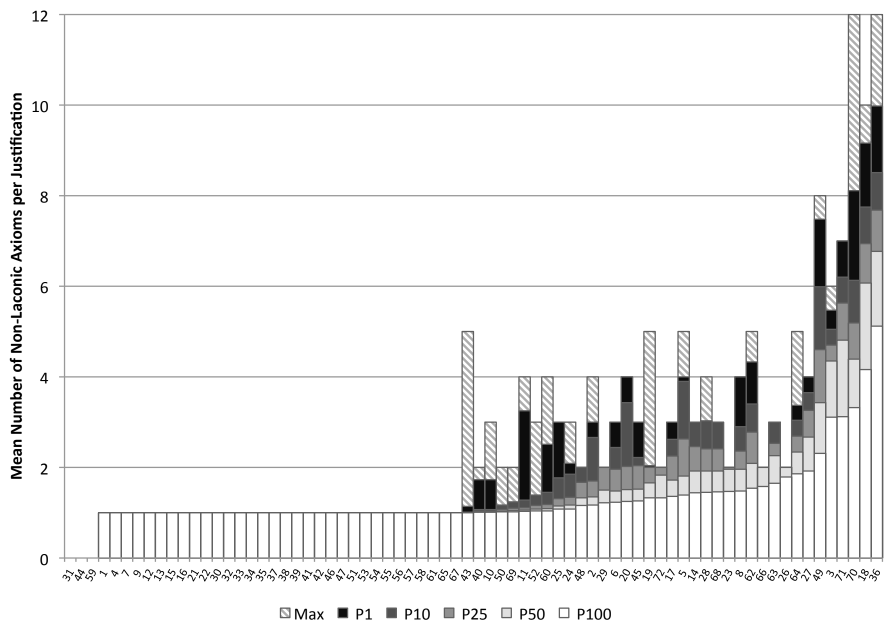


Figure 11.3: Experiment 4—The maximum and mean number of axioms containing superfluous parts per Justification. Means of percentiles are given. Pn represents the *top* n percent of justifications.

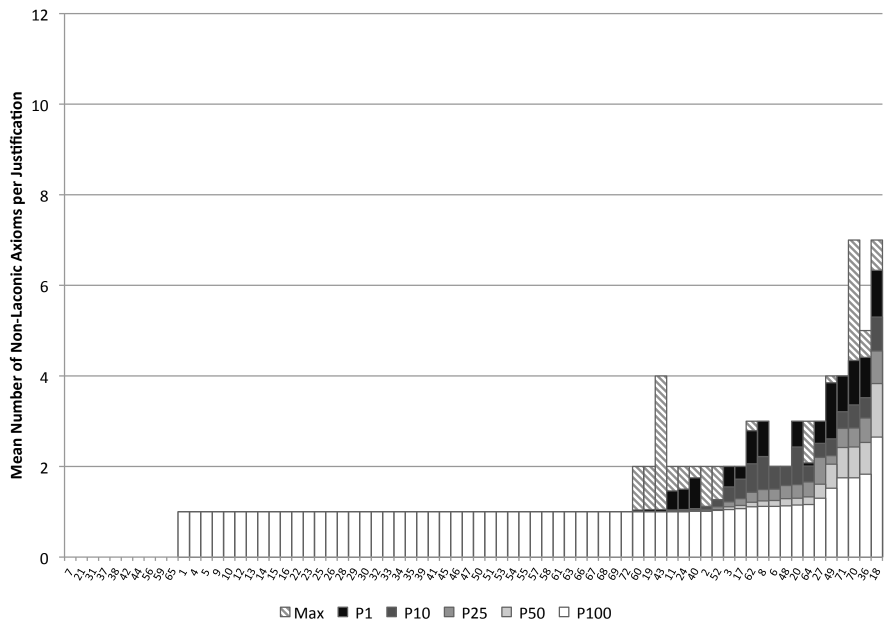


Figure 11.4: Experiment 4—The maximum and mean number of axioms containing superfluous parts per **Split** Justification. Means of percentiles are given. Pn represents the *top* n percent of justifications.

being in the order of hundreds of milliseconds rather than tens of seconds or hundreds of seconds. For most ontologies, 99 percent of justifications could be checked within 1,000 ms. Only Ontology 3 and Ontology 32 stand out as having an average runtime of over 1000 milliseconds. Ontology 3 is the amino-acid ontology, which has large justifications (an average of 25 axioms in size) with large axioms which results in $\delta(\mathcal{J})$ being large (on average 138 axioms in size). Ontology 32 is the imgt-ontology [GiL99], which Pellet (and FaCT++ and HermiT) struggles to perform entailment checking on in comparison to other ontologies. The high times here are due to high entailment checking times. The final ontology that stands out is Ontology 70, the uber-anatomy-ontology, which has a maximum laconic check time of 353,000 milliseconds. This is due to a combination of $\delta(\mathcal{J})$ size, which peaked at 162 axioms, and entailment checking performance which mounts up when checking this many axioms. However, in this particular ontology, cases like this are rare. Indeed 99 percent of the justifications can be checked well within 2,000 ms and 90 percent within 600 ms. In summary, the runtime performance of the implementation of Algorithm 10.1 seems to be acceptable to good for the purposes of checking real justifications in real ontologies. It must also be borne in mind that the implementation that was tested did not feature early termination as described in the listing of Algorithm 10.1 and so performance of the optimised algorithm is likely to be better than the tested implementation.

The Prevalence of Superfluity Figure 11.2 shows the percentage of justifications (per ontology) that are non-laconic. The first thing to notice is that justifications that are non-laconic are prevalent throughout the BioPortal corpus—over 50 percent of the non-split justifications in 59 of the 72 ontologies were non-laconic. Only three ontologies, 31 (47 justifications), 44 (48 justifications) and 59 (56 justifications) contained justifications all of which were laconic. As expected, there is a reduction in the prevalence of superfluity when concept equivalence axioms are split apart. Seven of the ontologies that contained laconic justifications do not contained any laconic justifications after splitting. This indicates that the superfluity in the non-split justifications was due purely to equivalent concept axioms where only one direction of the implication was needed. However, even with split axioms, it is still the case that over 50 percent of the justifications in 29 of the 72 ontologies are non-laconic, and out of the 72 ontologies there are still 61 ontologies that contain non-laconic split justifications.

Figure 11.3 shows the mean number of axioms in non-laconic non-split justifications with superfluous parts (the number of axioms that contribute to a justification being non-laconic), and Figure 11.4 shows the mean number of axioms in non-laconic split justifications by ontology. In many ontologies, the superfluity in non-laconic justifications, is confined to just one axiom (32 ontologies), however there are a significant number of ontologies where this is not the case. For example, Ontologies 3, 18, 36, 49 70 and 71 contain non-laconic justifications that have large numbers of non-laconic axioms in them. In particular, for Ontology 36, 50 percent of justifications have an average of just below 7 non-laconic axioms each, and 10 percent have an average of around 8.25, which a maximum of 12 axioms. After splitting, the average number of non-laconic axioms per justification drops somewhat—only 22 ontologies as opposed to 40 ontologies have more than one non-laconic axiom, but numbers are still high for the afore mentioned ontologies.

The Chances of Encountering a Non-Laconic Justification In summary, the chances of encountering a non-laconic justification for an entailment in the BioPortal corpus is rather high at 82.7%. The degree of and spread of superfluous parts over the axioms in a non-laconic justification varies quite a lot by ontology. This probably reflects the fact that different styles of modelling are present in different ontologies, and to some extent the degree of superfluity is influenced by modelling style.

11.2 Computing Preferred Laconic Justifications

In order to determine whether it is practical to compute preferred laconic justifications for entailments in real ontologies, the two optimised π based algorithms (Algorithm 10.7 and Algorithm 10.8) were implemented and tested. The details are presented in Experiment 5 and Experiment 6 and are described below.

Experiment 5: Algorithm 10.7 Empirical Evaluation

Experiment 5 Algorithm Implementation

Algorithm 10.7, the optimised algorithm for computing preferred laconic justifications using π, and its sub-routines, were implemented in Java.

Experiment 5 Test Data

The test data were a set of pairs. Each pair was an ontology from the set of BioPortal ontologies listed in Table 5.1, and a *filtered* set of the non-trivial direct atomic subsumptions and non-trivial direct class assertions that held in the ontology. The entailments sets were filtered by only including entailments for which it was possible to compute *all regular justifications* in Experiment 1. Further more, for reasons of practicalities, each entailment set was limited to 2000 entailments which were drawn from the full set using *random sampling*. This meant that the entailment sets of ontologies 16 (chemical information ontology), 19 (coriell cell line ontology), 21 (dermlex the dermatology lexicon), 26 (gene ontology extension), 36 (internaltional classification for nursing practice), 37 (international causes of external injuries), 38 (international classification of functioning disability and health icf), 43 (nci thesaurus), 60 (protein ontology) and 70 (uber anatomy ontology) had their entailment sets reduced to 2000 entailments. A random sample size of 2000 was chosen as, in all cases, it allowed conclusions to be generalised to the full set of non-trivial entailments with a margin of error of less than five percent (approximately 2%).

Experiment 5 Method

The implementation of Algorithm 10.7 was used to compute all preferred laconic justifications for each ontology/entailment pair in the input data. A time out of 10 minutes was imposed on computing all justifications, and a time out of 5 minutes was imposed on any one entailment check. The experiment run was conducted using the Pellet 2.2.2 reasoner, which was accessed via the OWL API. The experiments were performed on MacBook Pro with a 3.06 GHz Intel Core 2 Duo Processor. The Java Virtual Machine was allocated a maximum of 4 GB of RAM. For each entailment, the CPU time required to compute all regular justifications, and the CPU time required to compute all preferred laconic justifications was recorded.

Experiment 5 Results

Figure 11.5 shows the mean CPU time per entailment for computing all regular justifications (black bars) and the mean CPU time per entailment for computing all preferred laconic justifications (light coloured bars). The ontologies on the x-axis are ordered by mean CPU time required to compute all justifications. There were three ontologies for which Algorithm 10.7 failed to terminate within 10 minutes for *any* entailments. These were Ontology 43 (nci thesaurus), Ontology 45 (neural electro magnetic ontology), and Ontology 53 (ontology of clinical research), which appear on the right hand side as empty columns in Figure 11.5. Figure 11.6 shows the percentile times for computing all preferred laconic justifications in milliseconds.

Setting aside the three ontologies for which the implementation of Algorithm 10.7 failed to terminate, the mean time for computing preferred laconic justifications per entailment per ontology was 2,203 ms (SD=7,557 ms, Max=54,648 ms). This compares to a mean time for computing regular justifications per entailment per ontology of 668 ms (SD=2,151 ms, Max=13,036 ms).

Besides the three ontologies for which no preferred laconic justifications could be computed for any entailments, there were seven ontologies that contained *some* entailments for which not all laconic justifications could be computed. These ontologies and entailments are listed in Table 11.1, which shows the number of failures compared to the total number of entailments, and the average number of $\pi(\mathcal{O})$ justifications that were computed. The last column shows the number of entailments where not all preferred laconic justifications could be computed because entailment checking timed out. For example, in ontology 32, there were 35 failures, and all of these were due to entailment checking timeouts. Similarly, in Ontology 43 there were 2000 entailments and 2000 timeouts.

Experiment 5 Analysis

Overall, the performance of Algorithm 10.7 was good enough to be able to compute preferred laconic justifications for entailments in 69 out of the 72 ontologies. In 3 of the 72 ontologies there was, what can be best described as, catastrophic failure of the algorithm to compute any preferred laconic justifications for any entailments. This catastrophic failure is discussed below. For the remainder of the ontologies there were a handful of entailments for which not all preferred laconic justifications could be computed, but by and large, it was practical to compute all preferred laconic justifications for most entailments in the BioPortal ontologies using Algorithm 10.7.

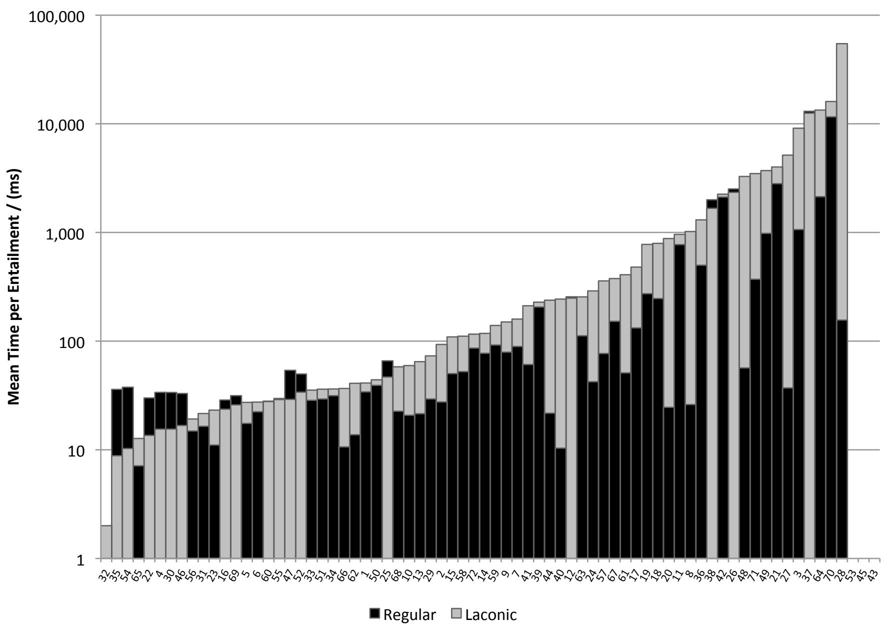


Figure 11.5: Experiment 5—Mean times to compute regular justifications versus preferred laconic justifications using Algorithm 10.7

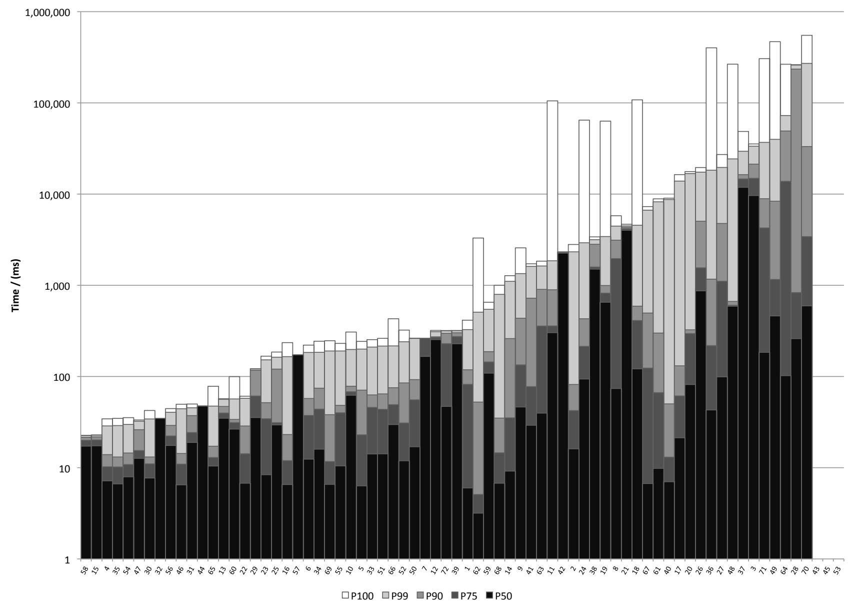


Figure 11.6: Experiment 5—Percentile times to compute preferred laconic justifications using Algorithm 10.7 (π based with top-level splitting). Ontologies (x-axis) are ordered by P99.

Table 11.1: π-Based Timeouts

Ontology	Failures	Mean Number of $\pi(\mathcal{O})$ Justifications Computed	Entailment Check Timeouts
2	1/91	152.0	0
18	5/345	276.6	0
19	1/2000	152.0	0
21	1/2000	0.0	1
32	35/36	0.0	35
36	3/2000	406.6	0
43	2000/2000	0.0	2000
45	139/139	0.0	139
49	7/703	578.1	0
53	99/99	0.0	99
64	5/547	25.0	3
70	8/2000	282.6	2

Reasons for Failures The reasons for the failure of Algorithm 10.7 to compute all preferred laconic justifications for an entailment can be boiled down into three main categories: (1) Failure due to large numbers of $\pi(\mathcal{O})$ justifications for an input $\mathcal{O} \models \eta$; (2) Failure due to a (catastrophic) degradation in entailment checking performance when checking entailments with respect to $\pi(\mathcal{O})$ in comparison to $\mathcal{O}$; and (3) A combination of (1) and (2). In the first case, the number of justifications for $\pi(\mathcal{O}) \models \eta$ can be (much) larger, and less overlapping (resulting in a blow up in the size of the hitting set tree) than the justifications $\mathcal{O} \models \eta$. This means that where it is possible to compute all justifications for η with respect to $\mathcal{O}$ it is not possible to compute all justifications for η with respect to $\pi(\mathcal{O})$. Another factor that comes into play is the degradation of entailment checking performance when checking entailments with respect to $\pi(\mathcal{O})$ rather than $\mathcal{O}$. Degradation occurs because $\pi(\mathcal{O})$ is larger (up to orders of magnitude larger) than $\mathcal{O}$ and, $\pi(\mathcal{O})$ typically contains more general concept inclusion axioms than $\mathcal{O}$. In combination, these two factors affect being able to compute all preferred laconic justifications. A mix of behaviour due to (1) and (3) was exhibited for all of the ontologies that did not fail catastrophically, with Ontology 7 being dominated by (1), and Ontology 64 being dominated by (2). For the ontologies that contained a handful of entailments for which not all preferred laconic justifications could be computed the number of $\pi(\mathcal{O})$ justifications that could be computed (of which some are preferred laconic justifications) tended to be large. This implies that "all is not lost" even though not all preferred laconic justifications could be computed.

Reasons for Catastrophic Failures There were three ontologies where the implementation of Algorithm 10.7 failed to compute *any* preferred laconic justifications for *any* entailments. These ontologies were Ontology 43 (nci thesaurus), Ontology 45 (neural electro magnetic ontology), and Ontology 53 (ontology of clinical research). For Ontology 43 and Ontology 53 the failures were due to the size of $\pi(\mathcal{O})$ when compared to the size of $\mathcal{O}$. Both of these ontologies contain axioms that have long and deeply nested class expressions. For example Ontology 53, contains the axiom

$$\begin{aligned}
\mathsf{HumanStudy} \equiv\ & \mathsf{OrganismalStudy} \\
& \sqcap (\exists \mathsf{entityDirectlyObserved}.(\mathsf{OCRE400076} \\
& \qquad \sqcup (\mathsf{OCRE400033} \sqcap \exists \mathsf{hasElements}.\mathsf{OCRE400076}) \\
& \qquad \sqcup \exists \mathsf{partOf}.\mathsf{OCRE400076})) \\
& \quad \sqcup (\exists \mathsf{entityRecruitedOrSelected}.(\mathsf{OCRE400076} \\
& \qquad \sqcup (\mathsf{OCRE400033} \sqcap \exists \mathsf{hasElements}.\mathsf{OCRE400076}) \\
& \qquad \sqcup \exists \mathsf{partOf}.\mathsf{OCRE400076})) \\
& \quad \sqcup (\exists \mathsf{entitySubjectToInterventionOrExposure}.(\mathsf{OCRE400076} \\
& \qquad \sqcup (\mathsf{OCRE400033} \sqcap \exists \mathsf{hasElements}.\mathsf{OCRE400076}) \\
& \qquad \sqcup \exists \mathsf{partOf}.\mathsf{OCRE400076}))
\end{aligned}$$

which results in a massive blowup of $\pi(\mathcal{O})$ for this ontology. Indeed, for this one axiom α, the size of $\pi(\{\alpha\})$ is over 47,000 axioms. The upshot of this explosion is that entailment checking performance degrades to the point where it makes in impractical to compute justifications with respect to $\pi(\mathcal{O})$ and hence it becomes impractical to compute preferred laconic justifications for any entailment in the whole ontology.

General Runtime Performance For the ontologies and entailments for which it was possible to compute all preferred laconic justifications the runtime performance of Algorithm 10.7 was arguably good and perfectly acceptable for use in a debugging and explanation service in an ontology development environment. Indeed, for most ontologies (65/69) the average time required to compute all preferred laconic justifications per entailment was less than 10 seconds, with 68/69 ontologies requiring less than 30 seconds on average and all 69 ontologies requiring less than 60 seconds on average. The percentile plot shown in Figure 11.6

gives a clearer picture as to the distribution of times—in particular the worst case times at the top end of the scale. While Ontology 28, has the largest average time per entailment, 75 percent of entailments actually fall below 1 second, with only 10 percent of entailments having times greater than 200 seconds. Looking at worst case times, only nine ontologies required greater than 100 seconds, and for seven of these, less than 1 percent of entailments per ontology required over this time. For all 69 ontologies, preferred laconic justifications could be computed for 75 percent of entailments in less than 60 seconds.

In Comparison to Computing Regular Justifications Out of the 72 ontologies, there were 18 ontologies where, on average, it took *less time* for computing *preferred laconic justifications* than it did to compute regular justifications. At this point, it is worth noting that in all of these 18 cases, the times for computing preferred laconic justifications were the same order of magnitude as the times for computing regular justifications. The main reason as to why it was quicker to compute preferred laconic justifications was due to the fact the there were fewer preferred laconic justifications in $\pi(\mathcal{O})$ that there were regular justifications in $\mathcal{O}$, and this was due to the effect of shared-core masking. Fewer justifications results in a smaller hitting set tree and better performance in computing all justifications.

There were 6 ontologies (Ontologies 8, 20, 27, 28, 40 and 48) where, on average, it took at least one order of magnitude more of time to compute preferred laconic justifications than it did to compute regular justifications. There main reasons for this is that for any one of these ontologies $\mathcal{O}$ the number of justifications contained in $\pi(\mathcal{O})$ is larger than the number of justifications contained in $\mathcal{O}$ (even though the number of preferred laconic justification might not be larger than the number of regular justifications for a given entailment). For example, in the case of Ontology 28, which had the largest time difference between time to compute regular and laconic justifications, the mean number of justifications per entailment in $\mathcal{O}$ was 4.56. However, the mean number of justifications per entailment with respect to $\pi(\mathcal{O})$ was 83.33 (compared with a mean number of preferred laconic justification per entailment of 12.56). In essence, for certain ontologies such as ones with deep subconcept nesting, the number of $\pi(\mathcal{O})$ can be much larger than the number of regular justifications or the number of preferred laconic justifications as $\pi(\mathcal{O})$ contains weakened justifications that lie between the regular justifications and preferred laconic justifications.

Based on the timing results from Experiment 6 it is clear that either the Algorithm can handle all (or most) entailments in an ontology, or it fails in spectacular fashion and cannot handle any entailments in an ontology. For ontologies where most or all entailments could be handled the timing results are encouraging. For most of these ontologies 99 percent of entailments can be handled in under one minute, and entailments in the other ontologies well within 10 minutes. This kind of performance is perfectly acceptable for the tasks of ontology debugging and repair. For the ontologies where the Algorithm failed to process any entailments, the incremental π based algorithm offers some hope of handling these ontologies and their entailments and this is borne out by the empirical evidence obtained in Experiment 6 below.

Experiment 6: Algorithm 10.8 Empirical Evaluation

Experiment 6 Algorithm Implementation

Algorithm 10.8, the optimised incremental algorithm for computing preferred laconic justifications based on π, and its sub-routines, were implemented in Java.

Experiment 6 Test Data

The test data were exactly the same as the test data used in Experiment 5.

Experiment 6 Method

The implementation of Algorithm 10.8 (computation of preferred laconic justifications through the incremental computation of $\pi(\mathcal{O})$) was used to compute all preferred laconic justifications for each entailment in each ontology/entailment set pair in the input data. A time out of 10 minutes was imposed on computing all justifications, and a time out of 5 minutes was imposed on any one entailment check. The experiment run was conducted using the Pellet 2.2.2 reasoner, which was accessed via the OWL API. The experiments were performed on MacBook Pro with a 3.06 GHz Intel Core 2 Duo Processor. The Java Virtual Machine was allocated a maximum of 4 GB of RAM. For each entailment, the CPU time required to compute all regular justifications, and the CPU time required to compute all preferred laconic justifications was recorded. Additionally, the number of regular and preferred laconic justifications per entailment was recorded.

Table 11.2: Incremental π-Based Timeouts

Ontology	**Failures**	**Mean Number of $\pi(\mathcal{O})$ Justifications Computed**	**Entailment Check Timeouts**
2	1/91	152.0	0
18	5/345	276.6	0
19	1/2000	152.0	0
32	30/36	0.0	30
36	3/2000	406.6	0
43	50/2000	0.0	2
45	6/139	0.0	0
49	9/703	578.1	0
53	3/99	0.0	0
64	5/547	25.0	3
70	8/2000	282.6	0

Experiment 6 Results

Figure 11.7 shows the mean CPU time in milliseconds for computing regular justifications and preferred laconic justifications per entailment per ontology. With the exception of Ontology 32, where there were failures for 30 out of 36 entailments, it was possible to compute preferred laconic justifications for most entailments in each of the 72 ontologies. Table 11.2 summarises the ontologies where failures occurred.

Figure 11.8 shows the percentile times for computing all justifications using the incremental π based algorithm. The ontologies are ordered by the 99th percentile times along the x-axis.

In order to draw out the differences between the incremental algorithm and non-incremental algorithm Figure 11.9 shows the ratio between the times for computing preferred laconic justifications using Algorithm 10.7 and using 10.8. For example, on average, Ontology 29 takes nearly two times as long to compute all preferred laconic justifications for an entailment using Algorithm 10.8 than it does using Algorithm 10.7. Since Ontology 32, 43 and 53 incurred catastrophic failures with Algorithm 10.7 they appear as empty bars on the right hand side.

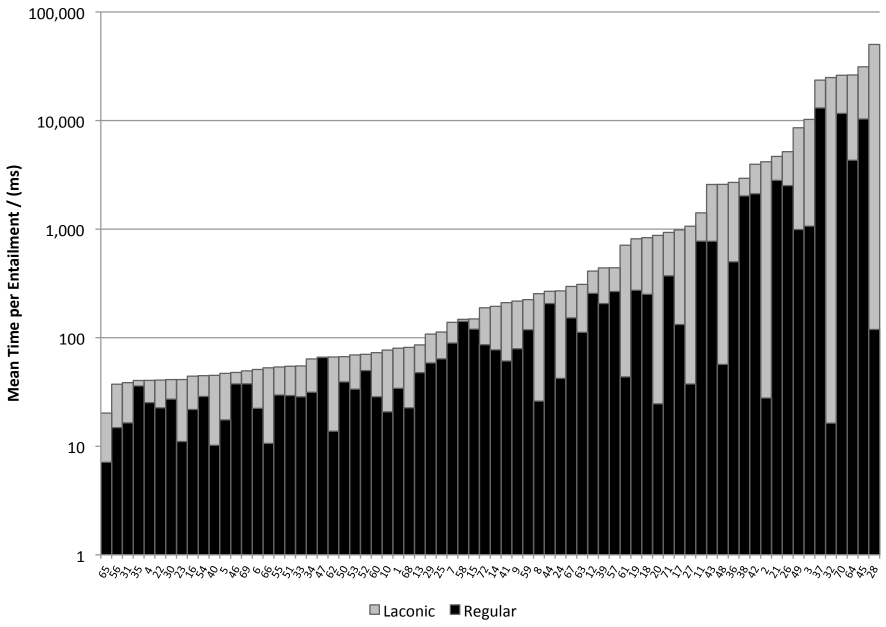


Figure 11.7: Experiment 6—Mean times to compute regular justifications versus preferred laconic justifications using Algorithm 10.8

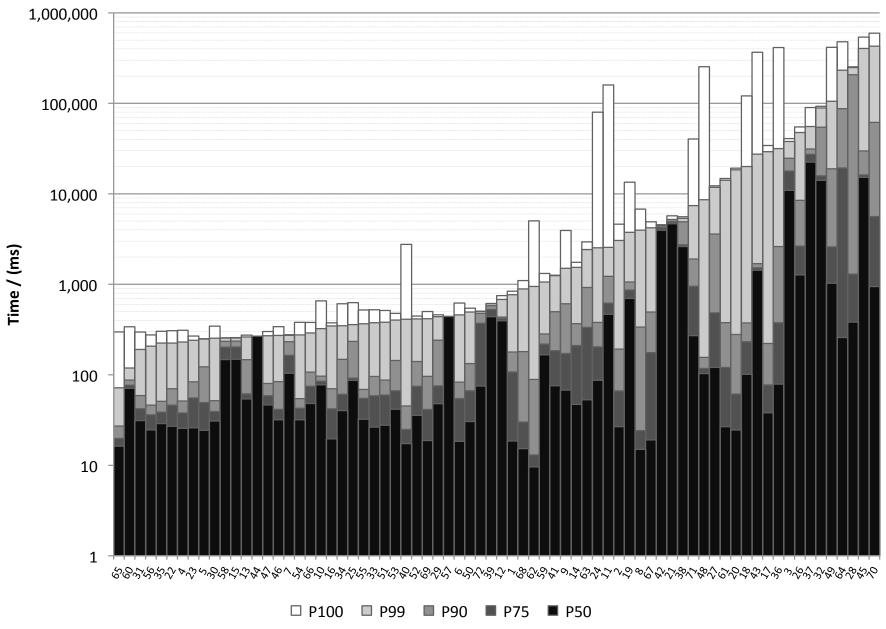


Figure 11.8: Experiment 6—Percentile times to compute preferred laconic justifications using Algorithm 10.8 (Incremental π based with top-level splitting). Ontologies (x-axis) are ordered by P99.

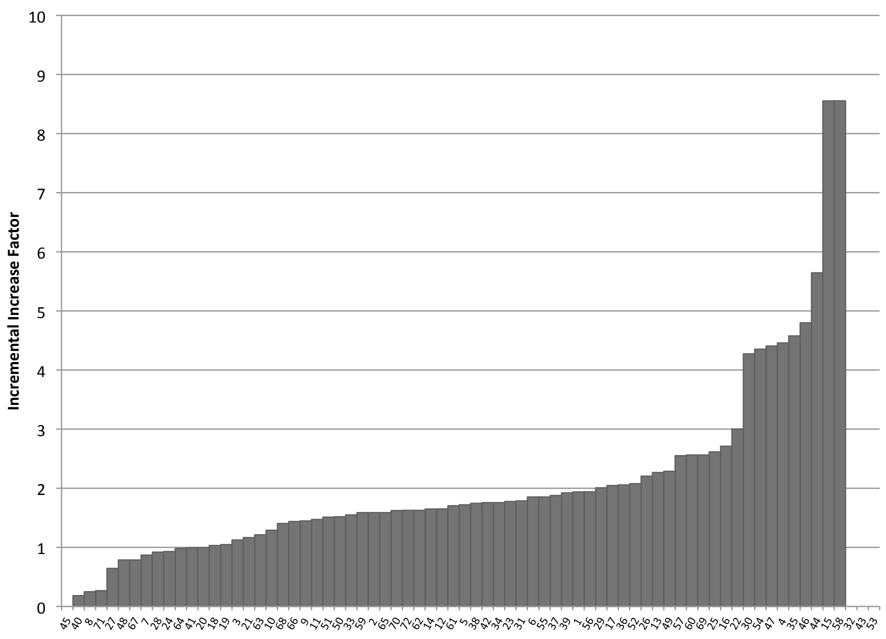


Figure 11.9: Experiment 6—Mean time ratio between Algorithm 10.7 and Algorithm 10.8

Experiment 6 Analysis

Overall Runtime Performance As with the previous algorithm, there are some failures to compute all preferred laconic justifications for some entailments, but there are no catastrophic failures this time. These failures and the robustness of the Algorithm 10.8 in comparison to 10.7 are discussed below. However, looking at Figure 11.7 and Figure 10.8 the overall performance of Algorithm 10.8 seems to be pretty good. For most ontologies the mean time was below 10 seconds and for all ontologies the mean time fell below 60 seconds. Looking at Figure 11.8, the worst case times and extremes can be seen. There are some entailments which require close to the 10 minute time out to compute all justifications (shown the right hand side of Figure 11.8), however as can be seen these fall between the 99th and 100th percentiles and thus represent the hardest 1 percent of entailments. In all but one ontology all justifications for 90 percent of entailments can be computed in less than 100 seconds, and for Ontology 28 where this is not the case, 90 percent of entailments can be fully dealt with within 200 seconds. Runtime performance of the algorithm is rather good, and is indicative that it could be used in practice in an ontology browsing and debugging environment.

In Comparison to Computing Regular Justifications As can be seen from 11.7, which shows the mean times for computing regular justifications compared to computing preferred laconic justifications using Algorithm 10.8, on average the time to compute laconic justifications is always greater. This is to be expected, since the Algorithm 10.8 requires at least two iterations of computing regular justifications over some set of axioms (the first being the ontology). Interestingly, the mean time for computing laconic justifications was not necessarily double the mean time for computing regular justifications on an ontology by ontology basis. The reason for this was due to the way that π performs top-level splitting, and in combination with shared core masking, the result is that, in some cases, entailment checking is faster in $\pi(\mathcal{O})$ than it is in $\mathcal{O}$. As a concrete example consider

$$\mathcal{O} = \{A \equiv B \sqcap C, A \equiv B \sqcap E, \ldots\} \models A \sqsubseteq B \ .$$

In $\pi(\mathcal{O})$, the entailment of interest $A \sqsubseteq B$ is exposed directly due to top level splitting of axioms in $\mathcal{O}$. The optimised justification finding algorithm (Algorithm 4.1) will determine that the entailment holds without having to load and query the reasoner. If $\mathcal{O}$ (or the module for $\mathcal{O}$) is several hundred axioms then the effect can become more pronounced. Additionally, in this example, shared core masking is present and $A \sqsubseteq B$ is the only preferred laconic justification, which also means runtime performance of Algorithm 10.8 is boosted over and above the performance of the regular justification finding algorithms with respect to $\mathcal{O}$.

The Performance of Algorithm 10.8 Compared with Algorithm 10.7 For entailments where it was possible to compute all preferred laconic justifications for both Algorithm 10.8 and Algorithm 10.7 the performance difference as a ratio between the mean times is summarised in Figure 11.9. As can be seen, there were no ontologies where there was over an order of magnitude difference in terms of time, although ontologies 15 and 59 come close at ratios of around 8.5. For most other ontologies the ratio fell below 2, and for all but four ontologies the ratio fell below 5. As explained above, the positive difference is to be expected, with times increasing over the base case of Algorithm 10.7 as the number of iterations increases (which occurs when external masking is present) and entailment checking is harder due to $\pi(\mathcal{O})$ being larger than $\mathcal{O}$.

The Robustness of Algorithm 10.8 compared with Algorithm 10.7 There were ten ontologies that contained several entailments for which it was not possible to compute all preferred laconic justifications using Algorithm 10.8. All 10 ontologies are ontologies which incurred failures with Algorithm 10.7—ontology-wise no new failures were introduced by use of Algorithm 10.8. At the level of entailments, many of the failed entailments (in ontologies 2, 18, 19, 36, 64, 70) were exactly the same as the failed entailments in Experiment 5, and was the algorithm failing due to the large numbers of justifications in $\pi(\mathcal{O})$ compared with the number of justifications in $\mathcal{O}$. Ontology 49 was the one ontology where there were two more failures with the incremental algorithm than there were with Algorithm 10.7, and these were due to the extra iterations in computing justifications that are required with the incremental algorithm—increasing the timeout from 10 minutes to 20 would get rid of most of these failures.

For Ontology 43, Ontology 45 and Ontology 53, the incremental algorithm does noticeably better than Algorithm 10.7, which failed to compute any preferred laconic justifications in these ontologies. Indeed, the incremental algorithm managed to compute all preferred laconic justifications for 1950/2000, 133/139 and 96/99 entailments in these ontologies respectively. This is obviously a huge improvement over the non-incremental algorithm. The reason for the improvement is that, as explained previously these ontologies contain axioms constructed from deeply nested complex concepts which has the effect of blowing up $\pi(\mathcal{O})$ and ruining entailment checking performance. With the incremental approach these axioms are only touched and brought into the subset of axioms from $\mathcal{O}$ that undergo the π transformation if parts of them appear in preferred laconic justifications. Since most of the sampled entailments in these ontologies have justifications that do not involved these highly nested complex concepts, the incremental algorithm performs better and does not fail in a catastrophic way.

11.3 The Laconic Justification Landscape

Figure 11.10 shows the mean number of regular and preferred laconic justifications per entailment per ontology. The ontologies along the x-axis are sorted by the ratio of the mean number of preferred laconic justifications to the mean number of regular justifications. There are 45 ontologies that, on average, have fewer laconic justifications than regular justifications per entailment. There are 8 ontologies that, on average, have more laconic justifications per entailment than regular justifications. In terms of masking, there were 9 ontologies that exhibited internal masking, 23 ontologies that exhibited external masking, and 53 ontologies exhibited shared core masking.

Figure 11.11 shows a bubble plot that depicts entailments where the number of regular justifications does not equal the number of preferred laconic justifications, and so gives some idea of masking over the whole BioPortal corpus. The x-axis shows the number of regular justifications and the y-axis shows the number of preferred laconic justifications. The size of the bubbles reflect the number of entailments with a particular ratio of regular to preferred laconic justifications. For example, the large bubble in the lower left corner represents entailments which have two regular justifications but only one preferred laconic justification, of which there are 5,447 entailments. The diagonal line which passes through centre of the graph from the lower left corner to the top right corner represents cases where the number of regular justifications is equal to the number of preferred laconic justifications. These instances are not shown on the graph. Bubbles which fall below this line represent entailments which definitely exhibit shared core masking, and bubbles which fall above the line represent entailments which definitely exhibit internal or external masking. Of course, entailments may exhibit shared core and internal or external masking is such ways that the number of regular and laconic justifications are equal, but these entailments are not represented on Figure 11.11.

Occurrences of Masking As can be seen from Figure 11.11 the phenomena of masking, be it internal, external or shared core masking is prevalent throughout the ontologies in the BioPortal corpus. Most of the ontologies (53 in total) exhibit shared core masking. This can be seen by looking from left to right in Figure 11.11, where the left most ontologies contain entailments with the highest ratio of regular to laconic justifications indicating a high degree of shared core masking. On the flip side of things, 9 ontologies exhibited external masking. The most extreme examples are shown from the right most column inwards in Figure 11.11, with Ontology 28 containing more preferred laconic justifications, on average, than regular justifications.

The bubble plot in Figure 11.11 shows where the vast majority of masking

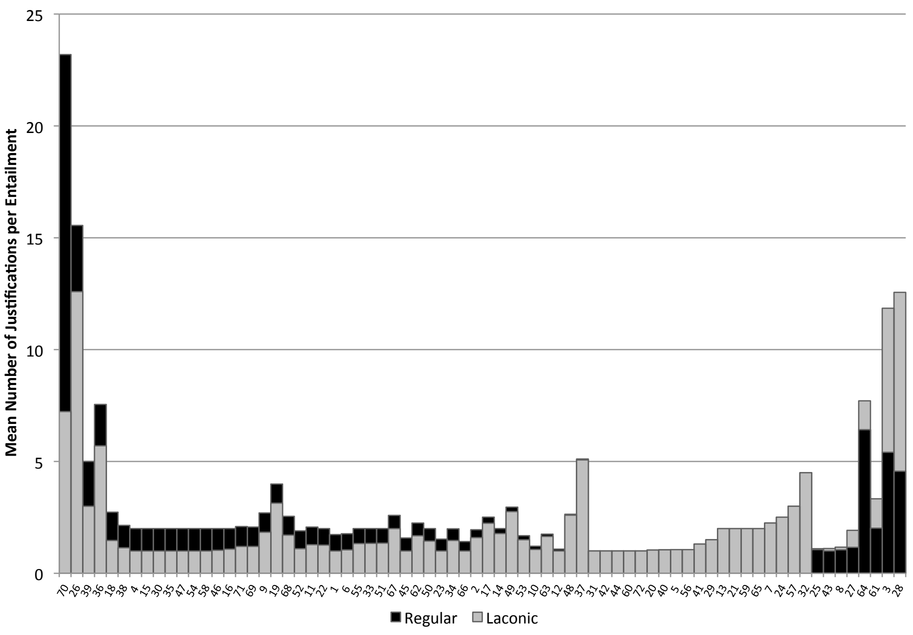


Figure 11.10: Mean number of regular and preferred laconic justifications per entailment

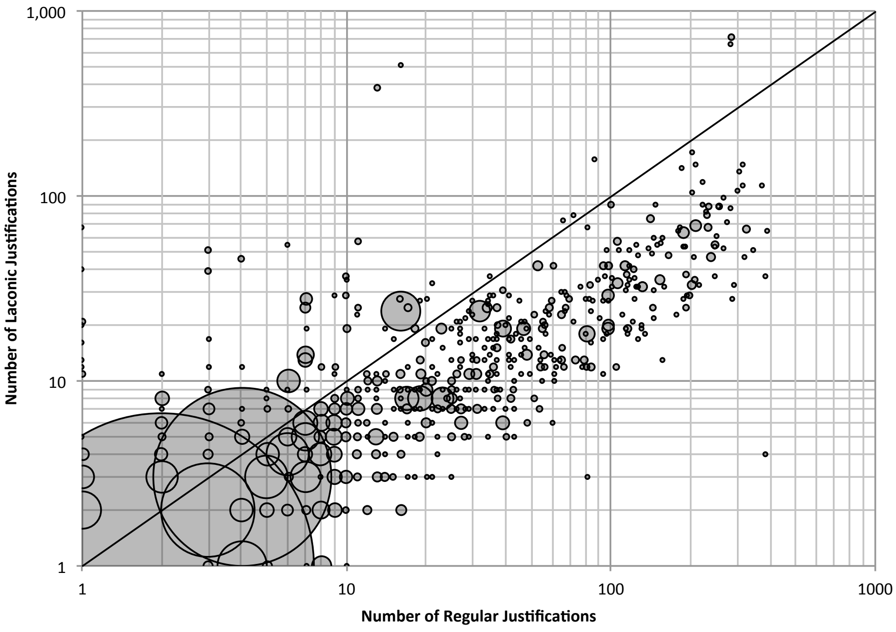


Figure 11.11: The Effect of Masking

cases lie[1]—shared core masking dominates, with plenty of entailment that have between 2 and 10 regular justifications but between 1 and 5 laconic justifications. However, there are also plenty of examples of external masking, with some of them being quite extreme. For example, the small bubbles that occur in the upper middle of Figure 11.11 represent entailments that have around 14-18 regular justifications but around 400 and 500 hundred laconic justifications. Even more extreme, the bubbles which lie on the y-axis represent entailments which have 1 regular justification but multiple laconic justifications. One of these extreme examples occurs in the NCI ontology which is discussed below.

An Example of Masking in the NCI Ontology The National Cancer Institute (NCI) Ontology is a huge ontology that contains around 146,000 logical axioms and has around 16,202 non-trivial entailments. Most of the justifications for these entailments are single axiom justifications (Mean size = 1, SD = 0.07, Max = 7) of the form $A \equiv B \sqcap C$ for the entailment $A \sqsubseteq B$ where C is a complex concept. An example of such a justification is:

$$\begin{aligned}\mathcal{J}_1 = \{\mathsf{ImmatureGastricTeratoma} \equiv & \mathsf{GastricTeratoma} \\ & \sqcap\ \mathsf{ImmatureExtragonadalTeratoma} \\ & \sqcap\ \mathsf{MalignantGastricGermCellNeoplasm}\}\end{aligned}$$

which is a single justification for the entailment $\mathsf{ImmatureGastricTeratoma} \sqsubseteq \mathsf{GastricTeratoma}$. At first sight, given that this is the only regular justification in this large ontology, the entailment does not look particularly non-trivial. However, the entailment has 4 preferred laconic justifications. The first is the entailment itself, which is obtained a weaker form of the regular justification after cutting out superfluous parts, i.e.

$$\mathcal{J}_1' = \{\mathsf{ImmatureGastricTeratoma} \sqsubseteq \mathsf{GastricTeratoma}\}$$

In this case, the superfluity is not so distracting that it makes the entailment hard to understand. However, there are three other externally masked justification in the ontology. The first, $\mathcal{J}_2'$, is 5 axioms in size and consists of a GCI that provide sufficient conditions for being a $\mathsf{GastricTeratoma}$ and two paths of subsumption axioms that traverse the class hierarchy.

[1]It should be noted that Figure 11.11 is a log-log plot and any bubbles which lie at a distance from the diagonal centre line represent fairly large differences.

$$\begin{aligned}
\mathcal{J}'_2 = \{&\mathsf{ImmatureGastricTeratoma} \sqsubseteq \mathsf{ImmatureExtragonadalTeratoma} \sqcap \mathsf{MalignantGastricGermCellNeoplasm}\\
&\mathsf{ImmatureExtragonadalTeratoma} \sqsubseteq \mathsf{ImmatureTeratoma}\\
&\mathsf{ImmatureTeratoma} \sqsubseteq \mathsf{Teratoma}\\
&\mathsf{MalignantGastricGermCellNeoplasm} \sqsubseteq \mathsf{GastricGermCellNeoplasm}\\
&\mathsf{GastricGermCellNeoplasm} \sqcap \mathsf{Teratoma} \sqsubseteq \mathsf{GastricTeratoma}\}
\end{aligned}$$

Two more externally masked justifications for this entailment are shown below as $\mathcal{J}'_3$ and $\mathcal{J}'_4$. Both justifications involved multiple definitions in the form of general concept inclusions (they are asserted into the ontology as equivalent concept axioms), and universal restrictions—they are non-trivial. Moreover, the justifications are of a reasonable size whose axioms are spread out over the ontology. It is doubtful that a person browsing or editing the ontology would realise that any of these extra axioms play a part in the entailment $\mathsf{ImmatureGastricTeratoma} \sqsubseteq \mathsf{GastricTeratoma}$ when confronted with the one and only regular justification, or indeed when simply manually browsing the ontology is a tool such as Protégé-4.

$$\begin{aligned}
\mathcal{J}'_3 = \{&\mathsf{ImmatureGastricTeratoma} \sqsubseteq \mathsf{ImmatureExtragonadalTeratoma} \sqcap \mathsf{MalignantGastricGermCellNeoplasm}\\
&\mathsf{ImmatureExtragonadalTeratoma} \sqsubseteq \mathsf{ImmatureTeratoma}\\
&\mathsf{ImmatureTeratoma} \sqsubseteq \mathsf{Teratoma}\\
&\mathsf{MalignantGastricGermCellNeoplasm} \sqsubseteq \mathsf{MalignantExtragonadalGermCellTumor} \sqcap \mathsf{MalignantGastricNeoplasm} \sqcap \forall \mathsf{hasSite}.\mathsf{Stomach}\\
&\mathsf{MalignantExtragonadalGermCellTumor} \sqsubseteq \mathsf{ExtragonadalGermCellNeoplasm}\\
&\mathsf{MalignantGastricNeoplasm} \sqsubseteq \mathsf{GastricNeoplasm}\\
&\mathsf{ExtragonadalGermCellNeoplasm} \sqcap \mathsf{GastricNeoplasm} \sqcap \forall \mathsf{hasSite}.\mathsf{Stomach} \sqsubseteq \mathsf{GastricGermCellNeoplasm}\\
&\mathsf{GastricGermCellNeoplasm} \sqcap \mathsf{Teratoma} \sqsubseteq \mathsf{GastricTeratoma}\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{J}_4' = \{&\mathsf{ImmatureGastricTeratoma} \sqsubseteq \mathsf{ImmatureExtragonadalTeratoma} \sqcap \mathsf{MalignantGastricGermCellNeoplasm}\\
&\mathsf{MalignantGastricGermCellNeoplasm} \sqsubseteq \mathsf{MalignantGastricNeoplasm} \sqcap (\forall \mathsf{hasSite.Stomach})\\
&\mathsf{MalignantGastricNeoplasm} \sqsubseteq \mathsf{GastricNeoplasm}\\
&\mathsf{ImmatureExtragonadalTeratoma} \sqsubseteq \mathsf{ImmatureTeratoma} \sqcap \mathsf{MalignantExtragonadalNonSeminomatousGermCellTumor}\\
&\mathsf{ImmatureTeratoma} \sqsubseteq \mathsf{Teratoma}\\
&\mathsf{MalignantExtragonadalNonSeminomatousGermCellTumor} \sqsubseteq \mathsf{MalignantExtragonadalGermCellTumor}\\
&\mathsf{MalignantExtragonadalGermCellTumor} \sqsubseteq \mathsf{ExtragonadalGermCellNeoplasm}\\
&\mathsf{GastricGermCellNeoplasm} \sqcap \mathsf{Teratoma} \sqsubseteq \mathsf{GastricTeratoma}\\
&\mathsf{ExtragonadalGermCellNeoplasm} \sqcap \mathsf{GastricNeoplasm} \sqcap \forall \mathsf{hasSite.Stomach} \sqsubseteq \mathsf{GastricGermCellNeoplasm}\}
\end{aligned}$$

Without the guidance of laconic justifications, not only would users of a tools be unaware of these reasons, which means that they do not fully understand the ontology, they might fail to repair the ontology if they hack at the axiom in the one and only regular justification and simply remove the GastricTeratoma conjunct for example.

As can be seen, external masking can be rather surprising when it arises in real, and especially large, ontologies. At first glance an ontology can look like it will only result in trivial inferences, but laconic justifications can peal back the superficial top layer and reveal the logical richness under the hood.

11.4 Discussion

Two important results emerge from the empirical evaluation detailed in this chapter. The first is that the phenomena of superfluity, internal masking, external masking and shared cores are prevalent in the BioPortal corpus. This indicates that the work is not aiming at a few isolated cases, and is likely to be of value to users who build and browse ontologies. The second is that it is largely practical to compute preferred laconic justifications for entailments in the BioPortal corpus ontologies. Without any further work on optimisations of the algorithms it is likely that the existing algorithms could be used in ontology development environments today.

Incremental versus Non-Incremental With regards to computing preferred laconic justifications, there were failures to compute all justifications for some entailments, and for the first algorithm (Algorithm 10.7) that was tested there were some catastrophic failures where no preferred laconic justifications could be computed for any entailments. However, the testing of the second algorithm (Algorithm 10.8) showed that it was possible to solve the problem of catastrophic failures by introducing an incremental approach that guards against blowups that the first algorithm is susceptible to, and confines failures to single entailments. Overall, in terms of performance there are benefits and trade offs between using Algorithm 10.7, the non-incremental algorithm, and Algorithm 10.8. The main benefit of using the incremental algorithm is that it is more robust in the face of ontologies which contain axioms that are built from highly nested complex concepts. In particular, it helps neutralise the effect of axioms that have absolutely nothing to do with an entailment making it impossible to compute justifications for that entailment. However, there is a price for this robustness in terms of time as the incremental algorithm requires multiple rounds of justification computation where as the non-incremental algorithm does not. As part of future work, it would be worth investigating the use of heuristics for choosing between the two algorithms based on input. It is possible that something rather simple would suffice, as the main influence in choice appears to be the degree of concept nesting in axioms.

11.5 Conclusions

The empirical results presented in this chapter provide strong evidence which indicates that:

- It is practical to detect whether or not realistic justifications are laconic.
- It is practical to compute all preferred laconic justifications for entailments in realistic ontologies. For 90% of direct atomic subsumptions and direct concept assertions type entailments in 71 out of 72 ontologies it was possible to compute all preferred laconic justifications within 60 seconds.
- There are entailments in realistic ontologies for which it is not possible to compute all preferred laconic justifications, but these are not particularly widespread—on average they account for less than 1.5% of entailments in 11 out of 72 ontologies.
- Superfluity and masking are widely exhibited in realistic ontologies. Out of the 72 BioPortal ontologies 53 exhibited masking, with 9 exhibiting internal

masking, 23 external masking and 53 shared-core masking. Over 82 percent of justifications from non-trivial entailments contained superfluity in their axioms.

Chapter 12

Understanding Justifications

As explained in Chapter 1, over the last few years justifications have become the dominant form of explanation in OWL ontology development environments and related tools. The main alternative to justifications as a form of explanation are proofs. Indeed, in some camps, proofs are regarded as the only real form of explanation [McG96, BFH00, Kwo05]. However, in comparison to full blown proofs, justifications are conceptually simple structures that have a direct bearing on what has been asserted or stated in an ontology. This means that, in order to understand how a justification works, it is not necessary to learn a proof calculus or a specific set of deduction rules—justifications do not require any additional knowledge beyond the semantics of the language. In essence, the conceptual simplicity of justifications, coupled with the fact that it is practical to compute them (Chapter 6) makes them a very attractive form of explanation.

However, despite this, casual observation of users working with justifications can reveal that there are naturally occurring justifications that people find difficult or impossible to understand. Indeed, when the justifications shown in Figure 12.1 and Figure 12.2, both from real ontologies, were presented to people, some of them had trouble trying to understand how each justification supports its entailment. Some people actually questioned whether the justification shown in Figure 12.1 was a justification at all.

In the case of the justification shown in Figure 12.1, which is a justification for $\mathsf{Person} \sqsubseteq \bot$, spotting that the justification entails $\top \sqsubseteq \mathsf{Movie}$ is key to understanding how the justification works. Since everything is entailed to be a Movie, and Person is disjoint with Movie, Person is disjoint with $\top$, hence Person is unsatisfiable. People who fail to realise that $\mathcal{J}_1 \models \top \sqsubseteq \mathsf{Movie}$ also generally fail to

$$
\begin{aligned}
\mathcal{J}_1 = \{&\mathsf{Person} \sqsubseteq \neg\mathsf{Movie}\\
&\mathsf{RRated} \sqsubseteq \mathsf{CatMovie}\\
&\mathsf{CatMovie} \sqsubseteq \mathsf{Movie}\\
&\mathsf{RRated} \equiv (\exists \mathsf{hasScript}.\mathsf{ThrillerScript}) \sqcup (\forall \mathsf{hasViolenceLevel}.\mathsf{High})\\
&\mathsf{Domain}(\mathsf{hasViolenceLevel},\ \mathsf{Movie})\}
\end{aligned}
$$

Figure 12.1: A justification for $\mathsf{Person} \sqsubseteq \bot$

understand how the justification gives rise to the entailment.

Similarly, the justification shown in Figure 12.2, is also difficult for people to understand. There are fifteen axioms of many different types, and it is far from obvious how these axioms interplay with each other to result in the entailment $\mathsf{Tabloid}(\mathsf{DailyMirror})$. When a person works through this justification, they have to spot intermediate entailments, for example, $\mathsf{WhiteVanMan}(\mathsf{Mick})$ and $\mathsf{Person}(\mathsf{Mick})$, in order to arrive at the conclusion $\mathsf{Tabloid}(\mathsf{DailyMirror})$.

A commonality between the two points detailed above is that subsets of a justification can result in entailments that can be viewed as “steps” or “intermediate entailments”. When trying to understand justifications it is necessary for people to spot and understand these intermediate entailments. The number of significant intermediate entailments, and for any given intermediate entailment, the number and types of axioms and concept expressions that give rise to the entailment, play an important part in dictating how difficult the justification is to understand.

Based on the fact that some naturally occurring justifications seem difficult to understand, this chapter investigates the understandability of justifications as forms of explanations. While there have been several user studies in the area of debugging [Lam07, KPSH05] and ontology engineering anti-patterns [RCVB09], there have not been any formal user studies that investigate the cognitive complexity of justifications. In what follows, the details of several user studies are presented, which show that there are naturally occurring justifications that can be very difficult to understand. Based on the results of these studies a simple complexity model which captures *some* of the aspects of justifications that make them difficult to understand is presented, and a methodology for iteration and refinement of the model is proposed. The next chapter provides an example of

$$
\begin{aligned}
\mathcal{J}_2 = \{&\mathsf{InverseProperties}(\mathsf{hasPet},\ \mathsf{isPetOf})\\
&\mathsf{isPetOf}(\mathsf{Rex}, \mathsf{Mick})\\
&\mathsf{Domain}(\mathsf{hasPet},\ \mathsf{Person})\\
&\mathsf{Male}(\mathsf{Mick})\\
&\mathsf{reads}(\mathsf{Mick}, \mathsf{DailyMirror})\\
&\mathsf{drives}(\mathsf{Mick}, \mathsf{Q123ABC})\\
&\mathsf{Van}(\mathsf{Q123ABC})\\
&\mathsf{Van} \sqsubseteq \mathsf{Vehicle}\\
&\mathsf{WhiteThing}(\mathsf{Q123ABC})\\
&\mathsf{Driver} \equiv \mathsf{Person} \sqcap \exists \mathsf{drives}.\mathsf{Vehicle}\\
&\mathsf{Driver} \sqsubseteq \mathsf{Adult}\\
&\mathsf{Man} \equiv \mathsf{Adult} \sqcap \mathsf{Male} \sqcap \mathsf{Person}\\
&\mathsf{WhiteVanMan} \equiv \mathsf{Man} \sqcap \exists \mathsf{drives}.(\mathsf{Van} \sqcap \mathsf{WhiteThing})\\
&\mathsf{WhiteVanMan} \sqsubseteq \forall \mathsf{reads}.\mathsf{Tabloid}\\
&\mathsf{Tabloid} \sqsubseteq \mathsf{Newspaper}\}
\end{aligned}
$$

Figure 12.2: A justification for Newspaper(DailyMirror)

how such a model can be used, in particular for the purposes of *justification lemmatisation* and *justification oriented proofs*, which are presented as mechanisms and structures that go beyond justifications, and which can be used to guide a person through a justification to reach the entailment that it supports.

12.1 User Studies

The initial motivation for this work was based on anecdotal evidence: the observation of people in OWL tutorials who were trying to understand justifications, and various posts on mailing lists by people asking for help in understanding justifications. In order to gain a more concrete feeling of what makes naturally occurring justifications difficult to understand, an exploratory user study was carried out. In the study, a cross section of people who work with OWL and ontologies were presented with a series of justifications and asked to explain why each justification resulted in the target entailment. The exploratory study is described below in Experiment 7. The principle goal was to gain qualitative data that could be used to bootstrap the construction of a justification complexity model. As well as characterising problematic features of justifications, a complexity model can be useful for application purposes such as spotting when an ontology might be difficult to maintain, deciding when to offer extra assistance to users trying to understand justifications, or for estimating where extra steps could be inserted into a justification in order to make it easier to understand.

Experiment 7: An Exploratory Study

Experiment 7 Participants

The study comprised 12 volunteers who were staff and students from the School of Computer Science at the University of Manchester. The participants' experience with OWL ranged from less than 6 months to over 4 years. Some participants only had experience in browsing ontologies, while other participants had developed OWL tools. All participants were either confident or very confident that given a rendering of an OWL axiom they could explain the meaning of the axiom to another person.

Experiment 7 Materials

A corpus of justifications for entailments found in published OWL ontologies was collected[1]. The entailments were either unsatisfiable class entailments ($A \sqsubseteq \bot$) or subsumption entailments between named concepts ($A \sqsubseteq B$). A subset of the corpus which speculatively seemed difficult for people to understand was selected (trivial justifications such as $\{A \sqsubseteq B \sqcap C\}$ as a justification for $A \sqsubseteq B$ were discarded). The subset of justifications was then *expanded*, using various substitution lemmas on the "difficult to understand" justifications, in order to provide a spectrum of justifications ranging from difficult to easy to understand. In total this provided a pool of 100 justifications which were used in the study.

Obfuscation In a series of dry runs, a biasing effect was noticed where domain experts in biology appeared to quickly understand the reason behind a fairly difficult justification obtained from the TAMBIS ontology [BGB$^+$99]. However,

[1]This exploratory study was carried out before the BioPortal became a mainstream repository for publishing ontologies. Indeed, BioPortal records indicate that it contained just 6 ontologies at the time of this experiment and none of these were the ontologies listed in Table 5.1 which contained non-trivial entailments. The corpus for this experiment was therefore found elsewhere.

after some discussion, it transpired that rather than "understanding the logic" the participant had simply used the names of classes as a cue, which coincidentally enabled them to pinpoint the reason for the unsatisfiable class. Therefore, for the real study, all justifications were *sanitised/obfuscated* by replacing the names of classes and properties in the signature of the justification with meaningless concept identifiers. `Cn` was used for concept names (where n represents a non-negative integer), and `prop`α (where α represents a letter A-Z) was used for roles.

Experiment 7 Method

A random selection of justifications was presented to each participant in a justification browser as an ordered/indented list of axioms. The ordering and indentation of axioms was generated automatically based on the signatures on the left and right hand side of general concept inclusion axioms. In some cases the ordering algorithm produces a less than desirable solution, therefore, the browsing tool allowed both the ordering and indentation of axioms to be modified at the wish of the participant. Finally, the tool allowed participants to switch between two different styles of syntax—the description logic style syntax, which is typically favoured by people with a logic background, and the *Manchester OWL Syntax* [HDG$^+$06, HPS08c], which is the syntax used in Protégé-4, OWLSight and Topbraid composer.

For each justification, the time taken for the participant to claim that they had understood (or had not understood) the justification was recorded. The "think-aloud protocol" [Lew82] was used in order for the study facilitator to determine whether or not the participant had, in fact, understood the justification. The participant's ranking on how easy or difficult the justification was to understand, was recorded using a six point *Likert* scale: {'Very easy'=1, 'Easy'=2, 'Neither easy or difficult'=3, 'Difficult'=4, 'Very difficult'=5, 'Impossible'=6}. A participant was free to stop the study at any time, and they were free to carry on ranking justifications for as long as they were comfortable doing so.

It should be noted that, at this stage, since it was not clear what features make justifications difficult understand, it was deemed better to get as many qualitative opinions and rankings on a wide range of justifications rather than show the same small number of justifications to every participant.

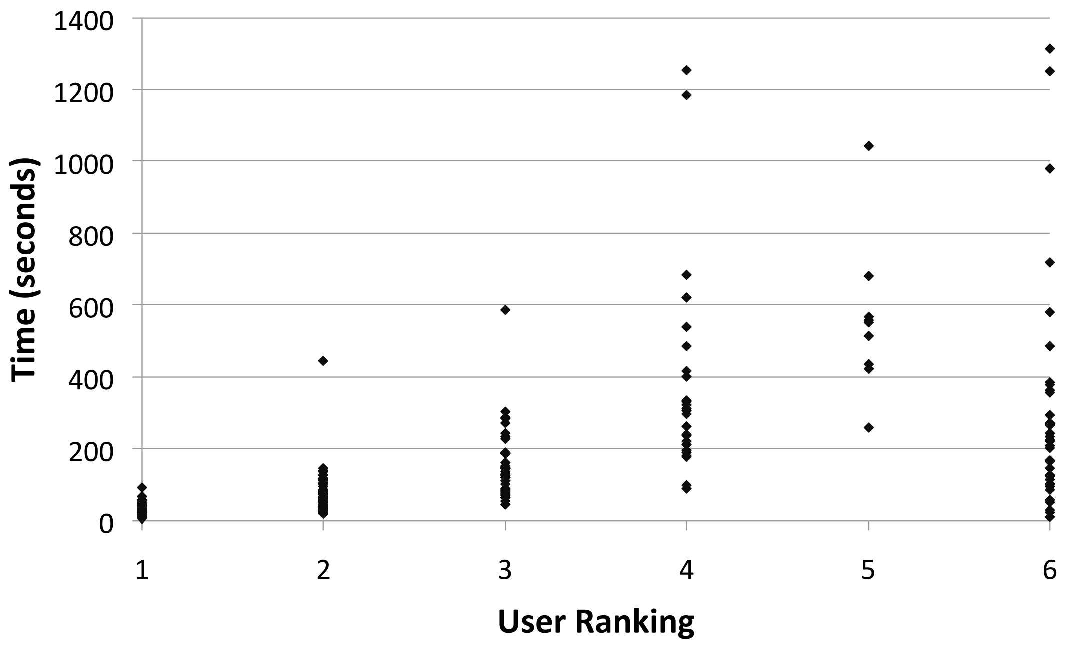


Figure 12.3: User Ranking versus Time

Experiment 7 Results

The total number of rankings was 227 (an average of 18.9 rankings per participant). Figure 12.3 shows a plot of ranking versus time (in seconds). Each point represents a single ranking. A rank of 1 corresponds to "very easy to understand", and a rank of 6 corresponds to "impossible to understand".

Out of the 227 rankings, 69 (30%) corresponded to being "difficult to understand" through to "impossible to understand" (35 rankings, (15%) corresponded to being "impossible" to understand). Interestingly, all of the "impossible to understand" rankings were rankings of unmodified, *naturally occurring*, justifications.

It was common for participants who could not understand a particular justification to ask the question, "Is this explanation correct?", thereby implying that they doubted the ability of the system to generate sound justifications. The initial appearance of a justification seemed important. Some people greeted certain justifications with shock and comments such as, "I'll never understand that". In some cases these expressions of immediate defeat proved correct and in other cases proved incorrect as people worked through these justifications to finally claim that they understood them.

Experiment 7 Analysis

Spread of Rankings As can be seen from Figure 12.3 there was a good spread of rankings over the justifications that received them. At both ends of the scale there were rankings for "very easy to understand" and "impossible to understand". Moreover, all participants bar one ranked one or more justifications as being impossible to understand. This one outlier participant did rank some justifications as being "very difficult to understand". Overall, there were a significant number of "difficult" to "impossible" to understand justifications, which points to the fact that justification understanding is a real problem.

Ranking versus Time Excluding ranking 6 ("impossible to understand") the general trend indicates that when participants took longer to understand a justification they perceived it to be more difficult to understand. It is noticeable that, in many cases the time spent trying to understand justifications that were deemed impossible to understand (ranking 6), is less than the time spent trying to understand very difficult justifications. This indicates that participants gave up trying to understand a justification very soon, perhaps because it simply seemed too complicated, or, after some effort they thought that they would never understand the justification.

The Number of Inference Steps When people work through justifications they typically perform obvious syntactic transformations, and recognise simple patterns, to reach intermediate conclusions. For example, consider the following justification:

$$\mathcal{J} = \{1\colon A \sqsubseteq B,\ 2\colon A \sqsubseteq \exists R.A,\ 3\colon D \equiv \exists R.B\} \models A \sqsubseteq D.$$

A person might work through this justification as follows: They spot that axioms 1 and 2 entail $A \sqsubseteq \exists R.B$, and then realise that this result, in conjunction with axiom 3 entails $A \sqsubseteq D$ (the entailment). Note how the user must spot a suitable intermediate inference step ($A \sqsubseteq \exists R.B$), understand how it arises, and understand the part it plays in the whole justification. The think-aloud protocol revealed that participants only saw some steps as being important. For example, people are not phased by large sets of simple axioms such as a long chain of concept subsumption axioms which they spot without working through each step in detail. As an illustration, consider the justification

$$\mathcal{J} = \{A \sqsubseteq B,\ B \sqsubseteq C,\ C \sqsubseteq D, D \sqsubseteq E,\ A \sqsubseteq \neg E\}$$

which entails $A \sqsubseteq \bot$, participants would not verbalise *every* step such as $A \sqsubseteq C$, but they would verbalise $A \sqsubseteq E$ and $A \sqsubseteq \neg E$. Justifications whose intermediate inference steps arise due to the interaction between many different types of axioms or concept expressions are hard to understand would cause people to verbalise more steps. Ultimately, justifications that contain a lot of information are difficult to understand. In particular, the number of different *types* of axioms and concept expressions that the justification contains seems to play an important part.

Patterns of Axioms Justifications that contain unfamiliar patterns of axioms are difficult to understand. Consider the following ontology, $\mathcal{O} = \{\alpha_1 \colon A \equiv \forall R.C$, $\alpha_2 \colon \mathsf{domain}(R, A)$, $\alpha_3 \colon E \sqsubseteq F\}$, which is derived from a real ontology[2] and was presented to some of the study participants as part of a larger justification. This ontology entails $E \sqsubseteq A$. The reason for this is that Axioms 1 and 2 entail $\top \sqsubseteq A$. During the study, it was observed that many of the participants (including participants with many years of experience with OWL, and even reasoner developers) did not realise, or neglected to see, that $A \equiv \forall R.C$, coupled with $\mathsf{domain}(R, A)$, entails $\top \sqsubseteq A$. Many of the participants had not encountered this "pattern of axioms" before. They therefore had difficulty in realising what these axioms entail, and their significance in the context of the complete justification. There are, of course, other patterns of axioms that occur in justifications that people find difficult to spot or understand.

12.2 A Simple Complexity Model

The data obtained from the user study provided an insight into how people read and tackle justifications, and why they find certain justifications difficult to understand. These insights, along with other basic intuitions, were used to develop a complexity model for predicting how complex a justification is to understand. Table 12.1 describes the model, wherein $\mathcal{J}$ is the justification in question, η is

[2]This example was taken from an ontology about movies, which was originally posted to the Protege-OWL mailing list and is the example shown in Figure 12.1. There are ontologies in the BioPortal corpus which contain similar patterns of axioms.

the focal entailment, and each value is multiplied by its weight and then summed with the rest. The final value is a complexity score for the justification. Broadly speaking, there are two types of components: (1) Structural components, such as **C1**, which require a syntactic analysis of a justification, and (2) Semantic components, such as **C4**, which require entailment checking to reveal non-obvious phenomena.

Components **C1** and **C2** count the number of different kinds of axiom types and class expression types as defined in the OWL 2 Structural Specification.[4] The more diverse the basic logical vocabulary is, the less likely that simple pattern matching will work and the more "sorts of things" the user must track.

Component **C3** detects the presence of universal restrictions where *trivial satisfaction* can be used to infer subsumption. Generally, people are often surprised to learn that if $\langle x, y\rangle \notin R^{\mathcal{I}}$ for all $y \in \Delta^{\mathcal{I}}$, then $x \in (\forall R.C)^{\mathcal{I}}$. This was observed repeatedly in the exploratory study.

Components **C4** and **C5** detect the presence of synonyms of $\top$ and $\bot$ in the signature of a justification where these synonyms are *not explicitly* introduced via subsumption or equivalence axioms. In the exploratory study, participants failed to spot synonyms of $\top$ in particular.

Component **C6** detects the presence of a domain axiom that is not paired with an (entailed) existential restriction along the property whose domain is restricted. This typically goes against peoples' expectations of how domain axioms work, and usually indicates some kind of non-obvious reasoning by cases. For example, given the two axioms $\exists R.\top \sqsubseteq C$ and $\forall R.D \sqsubseteq C$, the domain axiom is used to make a statement about objects that have R successors, while the second axiom makes a statement about those objects that do not have any R successors to imply that C is equivalent to $\top$. This is different from the typical pattern of usage, for example where $A \sqsubseteq \exists R.C$ and $\exists R.\top \sqsubseteq B$ entails $A \sqsubseteq B$.

Component **C7** measures maximum modal depth of sub-concepts in $\mathcal{J}$, which tend to generate multiple distinct but interacting propositional contexts.

Component **C8** examines the signature difference from entailment to justification. This can indicate confusing redundancy in the entailment, or synonyms of $\top$, that may not be obvious, in the justification. Both cases are surprising to people looking at such justifications.

Components **C9** and **C10** determine if there is a difference between the type

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

Table 12.1: Justification Complexity Model

Name	**Base value**	**Weight**
C1 AxiomTypes	Number of axiom types in $\mathcal{J}$ & η.	100
C2 ClassConstructors	Number of constructors in $\mathcal{J}$ & η.	10
C3 UniversalImplication	If an $\alpha \in \mathcal{J}$ is of the form $\forall R.C \sqsubseteq D$ or $D \equiv \forall R.C$ then 50 else 0.	1
C4 SynonymOfThing	If $\mathcal{J} \models \top \sqsubseteq A$ for some $A \in \mathsf{Signature}(\mathcal{J})$ and $\top \sqsubseteq A \notin \mathcal{J}$ and $\top \sqsubseteq A \neq \eta$ then 50 else 0.	1
C5 SynonymOfNothing	If $\mathcal{J} \models A \sqsubseteq \bot$ for some $A \in \mathsf{Signature}(\mathcal{J})$ and $A \sqsubseteq \bot \notin \mathcal{J}$ and $A \sqsubseteq \bot \neq \eta$ then 50 else 0.	1
C6 Domain&NoExistential	If $\mathsf{Domain}(R, C) \in \mathcal{J}$ and $\mathcal{J} \not\models E \sqsubseteq \exists R.D$ for some class expressions E and D then 50 else 0.	1
C7 ModalDepth	The maximum modal depth of all class expressions in $\mathcal{J}$.	50
C8 SignatureDifference	The number of distinct terms in $\mathsf{Signature}(\eta)$ not in $\mathsf{Signature}(\mathcal{J})$.	50
C9 AxiomTypeDiff	If the axiom type of η is not the set of axiom types of $\mathcal{J}$ then 50 else 0	1
C10 ClassConstructorDiff	The number of class constructors in η not in the set of constructors of $\mathcal{J}$.	1
C11 LaconicGCICount	The number of General Concept Inclusion axioms in a preferred laconic version of $\mathcal{J}$	100
C12 AxiomPathLength	The number of maximal length expression paths[3] in $\mathcal{J}$ plus the number of axioms in $\mathcal{J}$ which are not in some maximal length path of $\mathcal{J}$	10

of, and types of class expressions in, the axiom representing the entailment of interest and the types of axioms and class expressions that appear in the justification. Any difference can indicate an extra reasoning step to be performed by a person looking at the justification.

Component **C11** examines the number of subclass axioms that have a complex left hand side in a preferred laconic version of the justification. Complex class expressions on the left hand side of subclass axioms in a laconic justification indicate that the conclusions of several intermediate reasoning steps may interact.

Component **C12** examines the number of obvious syntactic subsumption paths through a justification. In the exploratory study, participants found it very easy to quickly read chains of subsumption axioms, for example, $\{A \sqsubseteq B, B \sqsubseteq C, D \sqsubseteq D, D \sqsubseteq E\}$ to entail $A \sqsubseteq E$. This complexity component essentially increases the complexity when these kinds of paths are lacking.

Model Tuning As can be seen, the model contains weights for various components. In should be noted that these are not definitive. They were determined by rough and ready empirical twiddling, without a strong theoretical or specific experimental backing. They correspond to observations made during the exploratory study of sufficient reasons for difficulty. The tuning was based on an iterative process: An initial set of weights was chosen based on the relative difficulty of complexity causing phenomena that were observed in the exploratory study. Based on these weights, the complexities for justifications from the pool used in the exploratory study were computed and then used to rank these justifications. Pairs of justifications were then eyeballed and the weights adjusted, if necessary, to alter the complexity scores and shift the position of justifications in the ranking. This process was repeated until what seemed to be a reasonable ordering on the justifications was achieved.

12.3 Model Validation

In what follows a series of model validation experiments are detailed. Before the experiments are described, it is worth noting that the experimental procedure used differs from that used in Experiment 7. The reason for this is that although the think-aloud protocol used in that experiment was robust and it worked very well, it was very resource intensive—it required a facilitator to sit with a participant throughout the whole experiment. This obviously prohibits the experiment being carried out remotely over the web for example, and limits the amount of data that can be collected over time. In order to remedy this, it was decided to design an experimental protocol that could be carried out without the need for one-to-one facilitation. To this end the experimental design is based on the methodology described by Newstead et al. in [NBH+06].

12.3.1 Experiment Design: Using Error Proportion to Indicate Difficulty

In [NBH+06] Newstead developed a model for predicting the difficult of logical reasoning problems used in the Graduate Record Examination (GRE)[5]. The basic approach followed by Newstead was to use error proportion and problem solving time as an indicator for the difficulty of a problem. That is, if a large proportion of participants give the incorrect answer to a question then that question contains features which make it difficult for the people to answer. Conversely, if a large proportion of participants give the correct answer to a question then that question is easy for the population to understand and answer.

In Newstead's experiments the questions, which are based on GRE questions, consist of scenarios with multiple choice questions based on those scenarios. Each question has one right answer and four wrong answers. It is therefore straight forward to assess whether or not a participant made an error in answering a question, and the notion of a participant making an error is therefore directly applicable to the kinds of questions that are asked in a GRE test. In terms of justifications, the notion of questions and errors is not so clear cut. Ideally, given a justification and a participant, an error is made if the participant fails to understand that justification. However, asking the question, "Do you understand this justification?" is undesirable. This is due to the qualitative nature of the question, which means that it can produce error proportions that do not reflect the true situation. This was evidenced in the exploratory study, where several participants claimed to have, or thought that they had, understood a justification but in actual fact they had not. In Experiment 7 these incidents were caught by use of the think-aloud protocol, but with a web-based, mass participant setup it is

[5]The Graduate Record Examination is a standardised test for admission into US universities and collages.

not possible to do verification like this. It was therefore necessary to approach the understanding justification task from a slightly different point of view. Instead of presenting a justification/entailment pair and asking a participant to try and understand the justification, it was decided to present a set of axioms/candidate pair and ask the participant to determine whether of not the candidate is entailed by the set of axioms. If the set of axioms *is* a justification, and the candidate entailment *is* the target entailment of that justification, then an error is made if the participant answers that the candidate is *not* entailed. While this setup differs from the standard justification situation, wherein the person looking at the justification knows it is a justification, it is still thought to be a reasonable proxy for the real task, and it provides a metric, in error proportion, that should relate well to justification complexity.

12.3.2 Justification Corpus

In order to thoroughly test the model it was decided to cast the net wide and randomly sample justifications from a selection of ontology corpora. Several well known ontology repositories were used: The Stanford BioPortal repository described in Chapter 5, the Dumontier Lab ontology collection[6] [MBB+01] (15 ontologies plus imports closure), the OBO XP collection[7] (17 ontologies plus imports closure) and the TONES repository[8] (36 ontologies plus imports closure). To be selected, an ontology had to (1) contain non-trivial entailments (Definition 5), (2) be downloadable and loadable by the OWL API (3) processable by FaCT++.

Although the selected ontologies cannot be said to generate a *truly representative* sample of justifications from the full space of possible justifications, they are diverse enough to put stress on many parts of the model. Moreover, most of these ontologies are actively developed and used and hence provide justifications that a significant class of users encounter.

For each ontology, the class hierarchy was computed, from which direct subsumptions between class names were extracted. For each direct subsumption, as many justifications as possible in the space of 10 minutes were computed—this typically meant that all justifications were computed, as timeouts were rare. This

[6]`http://dumontierlab.com/?page=ontologies`
[7]`http://www.berkeleybop.org/ontologies/`
[8]`http://owl.cs.manchester.ac.uk/repository/`

resulted in a pool of over 64,800 justifications.[9]

While large the pool of justifications is large in size, its actual logical diversity is considerably smaller. This is because many justifications, for different entailments, were of exactly the same "shape". For example, consider $\mathcal{J}_1 = \{A \sqsubseteq B, B \sqsubseteq C\} \models A \sqsubseteq C$ and $\mathcal{J}_2 = \{F \sqsubseteq E, E \sqsubseteq G\} \models F \sqsubseteq G$. As can be seen, there is an injective renaming from $\mathcal{J}_1$ to $\mathcal{J}_2$, and $\mathcal{J}_1$ is therefore *isomorphic* with $\mathcal{J}_2$. If a person can understand $\mathcal{J}_1$ then, with allowances for variations in name length, they should be able to understand $\mathcal{J}_2$. The initial large pool was therefore reduced to a smaller pool of 11,600 *non-isomorphic justifications.*

12.3.3 Experiment Setup

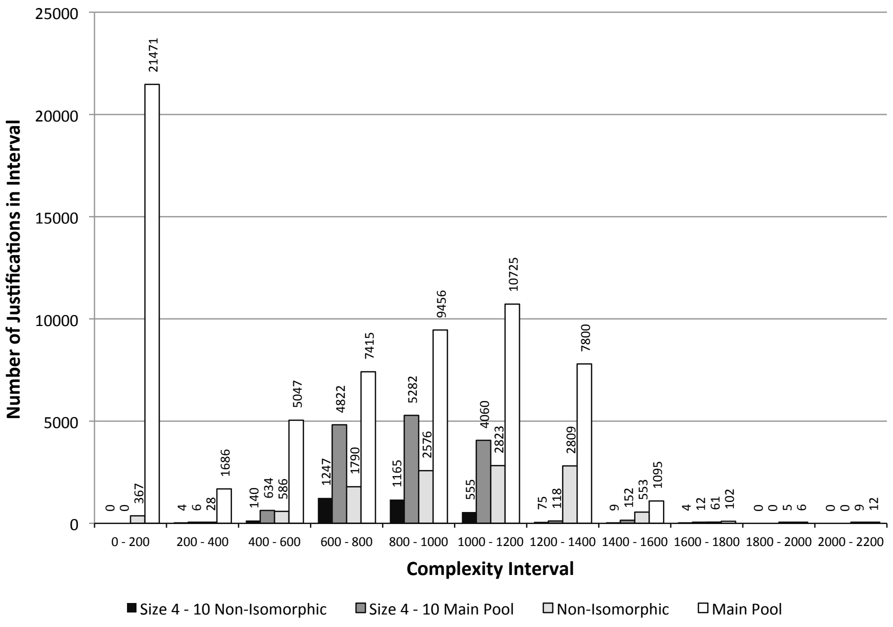
Items In the experiments that follow each experiment consists of a series of test *items* (questions from a participant point of view). A test *item* consists of a *set of axioms*, one *following axiom*, and a *question*, "Do these axioms entail the following axiom?". A participant *response* is one of five possible answers: "Yes" (it is entailed), "Yes, but not sure", "Not Sure", "No, but not sure", "No" (it is not entailed).

Item Construction For each experiment detailed below, test items were constructed from the pool of 11,600 non-isomorphic justifications. First, in order to reduce variance due primarily to size, justifications whose size was less than 4 axioms and greater than 10 axioms were discarded. This left 3199 (28%) justifications in the pool. In particular, this excluded large justifications that might require a lot of reading time, cause fatigue problems, and intimidate participants, and excluded very small justifications that tended to be trivial. Happily, it turns out that these bounds reflect the bounds of the mean sizes per ontology in the BioPortal corpus. One upshot of this size-based pruning is that nearly 40% of all justifications have no representative in the pruned set (see Figure 12.5). However, an inspection revealed that most of these were trivial single axiom justifications of the form $\{A \equiv B\} \models A \sqsubseteq B$ or $\{A \equiv (B \sqcap C)\} \models A \sqsubseteq B$, etc.

For each justification in the pool of the remaining 3199 non-isomorphic justifications, the complexity of the justification was computed according to the model

[9]It should be noted that this pool of justifications is considerably smaller than the pool obtained from the BioPortal in Chapter 3. The reason for this is that FaCT++ could not process some of the ontologies due to it not supporting certain datatypes—no attempt to repair any of the ontologies was made.

Figure 12.4: Justification Corpus Complexity Distribution



presented in Table 12.1, and then the justification was assigned to a complexity bin. A total of 11 bins were constructed over the range of complexity (from 0 to 2200), each with a complexity interval of 200. Next all bins which had zero non-isomorphic justifications of size 4-10 were discarded. This left 9 bins partitioning a complexity range of 200-1800. Figure 12.4 shows the overall picture of the initial state of the corpus and the effect of the reduction.

The final stage of item construction was justification obfuscation. All non-logical terms were replaced with generated symbols. Thus, there was no possibility of using domain knowledge to understand these justifications. The names were all uniform, syntactically distinguishable so that concept names appeared to be different to property names, and quite short. The entailment was the same for all items (i.e., $\mathsf{C1} \sqsubseteq \mathsf{C2}$).

Item Selection A key issue when designing user studies is to keep the length of the study to a minimum. This helps to ensure that participant fatigue does not set in during the study which means that participants are less likely to give up early or guess at answers simply to end the study as soon as possible. Based on the length of time that people spent examining justifications in the exploratory study it was decided to limit the number of items to 6 and to focus on a "easy/hard" divide of the lowest three non-empty bins (200-800) and the highest three non-empty bins (1200-1800).

Choosing justifications like this, from either end of the spectrum, has some consequences. In particular, a lot of the justifications, both with and without isomorphic reduction, lie in the middle of the spectrum, and these justifications are not sampled from. However, there were two good reasons for ignoring these middle ground justifications: (1) It is not clear how granular or sensitive the model is, and whether a difference in score of around 400 between two justifications $\mathcal{J}_1$ and $\mathcal{J}_2$ would really mean $\mathcal{J}_1$ is significantly easier to understand than $\mathcal{J}_2$; and (2) Error proportion is being used to distinguish easy from difficult to understand justifications. However, it is not clear whether the variance in participant error proportion would be low enough to produce a meaningful difference for justifications that are separated by relatively small model scores. While choosing justifications from each end of the spectrum limits the claims than can be made about model performance over the entire corpus, it does strengthen negative results in that if the model cannot distinguish the two poles, where the largest effect is expected to be, then either the model has failed or error proportions are not a reliable marker for difficulty.

Item Makeup: Justifications versus Non-Justifications From a participant point of view, an item may or may not represent a justification. However, in the actual experiments which follow, every item was in fact a justification—there were no non-entailing sets of axioms that were presented. The reasons for this choice were: (1) It maximises the number of real justifications examined; (2) Justification understanding is the actual task at hand, and it is not clear how much non-entailment checking would distort things and whether or not it would be much harder than checking for entailment; and (3) It is unclear how to interpret error rates for non-entailments in light of the model—the model predicts the difficulty of justifications, not non-entailing sets of axioms.

12.3.4 The Experiments

Given the basic structure of an experiment, in terms of items and setup, each experiment is now presented in detail. Three experiments were carried out: (1) A Pilot Study, which aimed to test whether error proportion is a satisfactory predictor of difficulty, and to verify the experimental setup; (2) A repeat of the pilot study with volunteers from a pool of MSc students; and (3) A followup experiment to check and explain outcomes from the second experiment.

Experiment 8: Pilot Study

Experiment 8 Participants

Seven members of the School of Computer Science at the University of Manchester. Participants were either Academic, Research Staff, or PhD Students, with over 2 years of experience with ontologies and justifications.

Experiment 8 Materials and procedures:

The study was performed using an in house, web based survey tool. The tool tracks times between all clicks on the page and thus records the time to make each decision.

The participants were given a series of test items consisting of 3 practice items, followed by 1 easy item (**E1** of complexity 300) and then four additional items, 2 ranked easy (**E2** and **E3** of complexities 544 and 690, resp.) and 2 ranked hard (**H1** and **H2** of complexities 1220 and 1406), which were randomly ordered for each participant. The easy items were drawn from bins 200-800, and the hard items from bins 1200-1800. The expected time to complete the study was a maximum of 30 minutes, including the orientation, practice items, and brief demographic questionnaire (taken after all items were completed).

Experiment 8 Results:

Errors and times are given in Table 12.2. Since all of the items were in fact justifications, participant responses were recoded to success or failure as follows: Success = ("Yes" | "Yes, but not sure") and Failure = ("Not sure" | "No, Not sure" | "No"). Error proportions were analysed using a Cochran Q Test [Coc50][10], which takes into consideration the pairing of successes and failures for a given participant. Times were analysed using two tailed paired sample t-tests.

[10]A Cochran Q Test [Coc50] is a non-parametric test of whether k treatments, with success or failure outcomes, have identical effects. It is essentially a McNemar [McN47] test which is applied to three or more groups.

Table 12.2: Pilot Study Items

Item	**Failures**	**Mean Time (ms)**	**Time S.D. (ms)**
E1	0	65,839	39,370
E2	1	120,926	65,950
E3	2	142,126	61,771
H1	6	204,257	54,796
H2	6	102,774	88,728

An initial Cochran Q Test across all items revealed a strong significant difference in error proportions between all items [$Q(4) = 16.00$, $p = 0.003$]. Further analysis using Cochran's Q Test on pairs of items revealed strong statistically significant differences in error proportion between: **E1/H1** [$Q(1) = 6.00, p = 0.014$], **E1/H2** [$Q(1) = 6.00$, $p = 0.014$] **E2/H2** [$Q(1) = 5$, $p = 0.025$] and **E3/H2** [$Q(1) = 5.00$, $p = 0.025$]. The differences in the remaining pairs, while not exhibiting differences above $p = 0.05$, were quite close to significance, i.e., **E2/H1** [$Q(1) = 3.57$, $p = 0.059$] and **E3/H1** [$Q(1) = 2.667$, $p = 0.10$].

An analysis of times using paired sample t-tests revealed that time spent understanding a particular item is not a good predictor of complexity. While there were significant differences in the times for **E1/H1** [$p = 0.00016$], **E2/H1** [$p = 0.025$], and **E3/H1** [$p = 0.023$], there were no significant differences in the times for **E1/H2** [$p = 0.15$], **E2/H2** [$p = 0.34$] and **E3/H2** [$p = 0.11$].

Experiment 8 Summary

In summary, these significant differences in error proportions were encouraging and provided an indication that error proportions and the experimental setup were performing as expected. That is, high model scores indicate difficult justifications which cause people to make errors in the experiment, and low model scores indicate easy justifications on which people do not make errors. With regards to times, the result of there being no significant differences between easy and hard justifications was anticipated. As in the exploratory study people gave up very quickly for justifications that they felt they could not understand. In essence, time is not a good indicator of difficulty. Finally, it is worth noting that all participants, including participants with a background in OWL and Description Logics made errors on the predicted hard justifications.

The pilot study proved successful in showing that error proportions can be used to distinguish between easy and hard justifications. The basic protocol was therefore taken forward and the experiment was repeated on a volunteers from a class of MSc students.

Experiment 9: MSc Student Cohort

Experiment 9 Participants

14 volunteers from a Computer Science MSc course on OWL ontology modelling. Participants were given chocolate in return for their participation. Each participant had minimal exposure to OWL (or logic) before the MSc course, but had, over the 5 week duration of the course, constructed and manipulated several ontologies, and received an overview of the basics of OWL and reasoning, etc. However, they did not receive any specific training on justifications.

Experiment 9 Materials

The study was performed according to the protocol used in the pilot study, but a new set of items was used. Since the mean time taken by pilot study participants to complete the survey was 13.65 minutes, with a standard deviation of 4.87 minutes, an additional hard justification was added to the test items. Furthermore, all of the items with easy justifications ranked easy were drawn from the highest easy complexity bin (bin 600-800). Results from the pilot study indicated that the lower ranking easy items were found to be quite easy and an inspection of their bins indicated that another draw would result in a similar set of justifications. It was therefore decided to draw an extra justification from the third bin (600-800) which is much larger, more logically diverse, and is therefore more challenging for the model. The series consisted of 3 practice items followed by 6 additional items, 3 easy items (**EM1**, **EM2** and **EM3** of complexities: 654, 703, and 675), and 3 hard items (**HM1**, **HM2** and **HM3** of complexities: 1380, 1395, and 1406).

Experiment 9 Method

The items were randomly ordered for each participant. Again, the expectation of the time to complete the study was a maximum of 30 minutes, including orientation, practice items and brief demographic questionnaire.

Table 12.3: MSc Student Cohort Results

Item	Failures	Mean Time (ms)	Time S.D. (ms)
EM1	6	103,454	68,247
EM2	6	162,928	87,696
EM3	10	133,665	77,652
HM1	12	246,835	220,921
HM2	13	100,357	46,897
HM3	6	157,208	61,437

Experiment 9 Results

Errors and times are presented in Table 12.3. The recoding of answers into success of failure is the same as in the pilot study. An analysis with Cochran's Q Test across all items reveals a significant difference in error proportion over all items [$Q(5) = 15.095, p = 0.0045$].

A pairwise analysis between easy and hard items reveals that there are significant and, highly significant, differences in errors between **EM1/HM1** [$Q(1) = 4.50$, $p = 0.034$], **EM1/HM2** [$Q(1) = 7.00$, $p = 0.008$], **EM2/HM1** [$Q(1) = 4.50$, $p = 0.034$], **EM2/HM2** [$Q(1) = 5.44$, $p = 0.02$], and **EM3/HM2** [$Q(1) = 5.44$, $p = 0.02$].

However, there were no significant differences between **EM1/HM3** [$Q(1) = 0.00$, $p = 1.00$], **EM2/HM3** [$Q(1) = 0.00$, $p = 1.00$], **EM3/HM3** [$Q(1) = 2.00$, $p = 0.16$] and **EM3/HM1** [$Q(1) = 0.67$, $p = 0.41$].

Experiment 9 Summary

In line with the results from the pilot study, an analysis of times using a paired samples t-test revealed significant differences between some easy and hard items, with those easy times being significantly less than the hard times **EM1/HM1** [$p = 0.023$], **EM2/HM2** [$p = 0.016$] and **EM3/HM1** [$p = 0.025$]. However, for other pairs of easy and hard items, times were not significantly different: EM1/HM1 [$p = 0.43$], **EM2/HM1** [$p = 0.11$] and **EM3/HM2** [$p = 0.10$]. Again, time is not a reliable predictor of model complexity.

With regards to the nonsignificant differences between certain easy and hard items, there are two items which stand out: An easy item **EM3** and a hard item **HM3**. The easy item cause more errors than would be expected by the model score, and the hard item vice-versa.

For item **EM3**, a plausible explanation for this is that a certain pattern of superfluous axiom parts in the item, which were not recognisable by the model, made it harder than the model predicted. That is, that the *model* was wrong.

For item **HM3** the explanation is a little different. Justifications similar to the justification in **HM3** have been observed to stymie experienced modellers in the field. Furthermore, it involves deriving a synonym for $\top$, which was not a move this cohort had experience with, but from the think-aloud protocol in exploratory study is known to cause problems. A plausible explanation for the fact that the MSc students answered "Yes" is that a misleading pattern of axioms in the first and last axioms were present in item **HM3**. The high "success" rate was therefore due to an error in reasoning, that is, a *failure* in understanding rather than a failure in the model—the MSc students got the answer right but for the wrong reasons.

In order to determine the plausibility of the above conjectures a follow up experiment was conducted, with the goal of observing the conjectured behaviours *in situ*. Although this experiment cannot explain exactly what happened in Experiment 9, it can verify that the conjectured behaviour occurs in practice.

Experiment 10: Anomaly Investigation with Think-Aloud

Experiment 10 Participants

Two CS Research Associates and one CS PhD student none of whom had taken part in the pilot study. All participants were very experienced with OWL.

Experiment 10 Materials

Items and protocol were exactly the same as Experiment 9

Experiment 10 Procedure

The same as 9 with the addition of the think-aloud protocol. An eye tracker was used throughout the experiment which also had the facility to record sound and the movements of the mouse pointer over the screen.

Experiment 10 Results

Figure 12.5 shows an eye tracker heat map for the most extreme case of distraction in item **EM3**. Think-aloud revealed that all participants were initially fixated on trying to figure out what role the $\exists\,\mathsf{prop1.C6}$ conjunct in the third axiom played in the justification. All of them were distracted by this superfluous conjunct and struggled with the justification to begin with. However, in the end, unlike the MSc students, all participants gave the correct answer and no errors were made. In the case of **HM3**, think-aloud revealed that none of the participants understood how the entailment followed from the set of axioms. However, two of them responded correctly and stated that the entailment did hold.

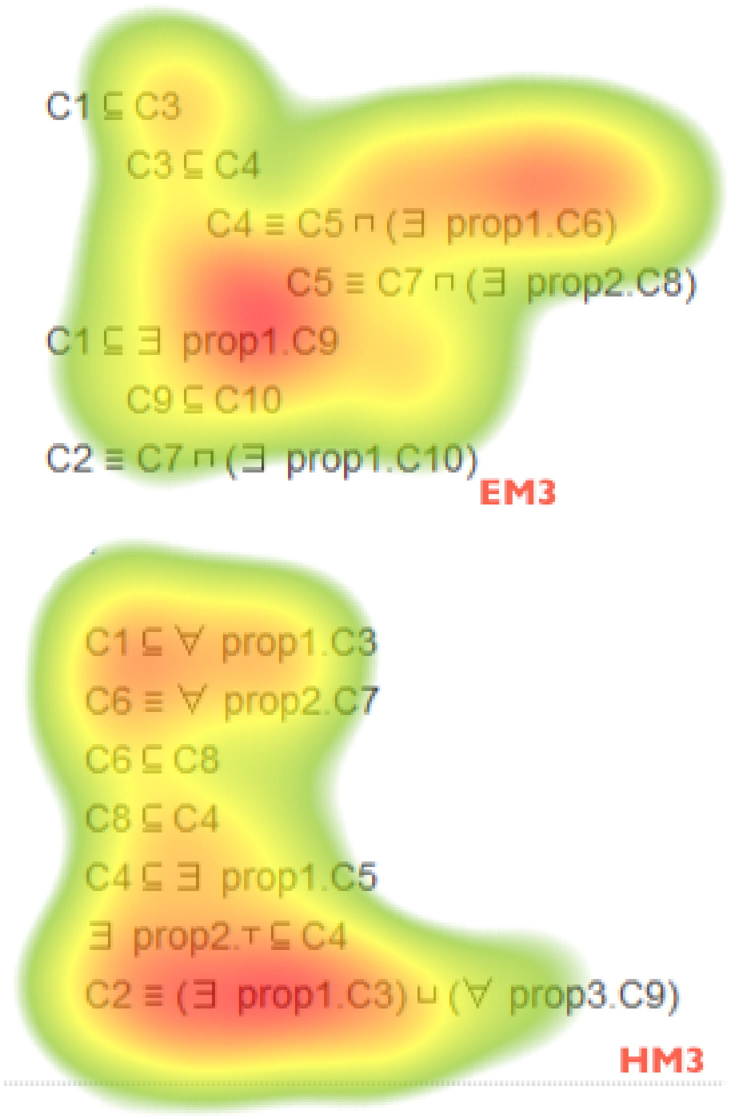
Experiment 10 Summary

With regards to **EM3**, think-aloud revealed that all participants were distracted by the superfluous axiom parts in item **EM3**. As can be seen, hot spots lie over the superfluous parts of axioms. While all of the participants did answer correctly in the end and, (as predicted by the model there were no failures) the superfluous parts did have a negative impact on the reading of the justification—all of them struggled with it at first. It is conceivable that these distracting parts had more of a negative impact on the MSc students, who were not as experienced and not confident enough to read through them and come to the conclusion that the entailment did indeed hold.

With regards to **HM3**, as conjectured, the patterns formed by the start and end axioms in the item set seemed to mislead participants. In particular, when disregarding quantifiers, the start axiom $\mathsf{C1} \sqsubseteq \forall\,\mathsf{prop1.C3}$ and the end axiom $\mathsf{C2} \sqsubseteq \exists\,\mathsf{prop1.C3} \sqcup \ldots$ look very similar. One participant spotted this similarity and claimed that the entailment held as a result. The focus on these start and end axioms is evident from Figure 12.5, where hot spots occur over the final axiom and the first axiom, with relatively little activity in the axioms in the middle of the justification. This tallies with research on item ordering [Pot74], which confirms that people pay more attention to items at the beginning and ends of lists.

In summary, the eye tracking and think-aloud experiment findings align with explanations for the anomalies in Experiment 9. That is, superfluity in justifications is distracting and caused people to struggle with **EM3**, and people got **HM3** right for the wrong reasons.

Figure 12.5: Eye Tracker Heat Maps for **EM3** & **HM3**



12.4 Dealing with Justification Superfluity

Perhaps the biggest issue with the current model is that it does not deal at all with superfluity in axioms in justifications. That is, it does not penalise a justification for having axioms that contain, potentially distracting, superfluous parts—parts that do not matter as far as the entailment is concerned. Unfortunately, without a deeper investigation, it is unclear how to rectify this in the model. Although it is possible to identify the superfluous parts of axioms using Laconic and Precise Justifications, throwing a naive superfluity component into the model would quite easily destroy it. This is because there can be justifications with plenty of superfluous parts that are trivial to understand. For example consider $\mathcal{J} = \{A \sqsubseteq B \sqcap C\} \models A \sqsubseteq B$, where C is along and complex class expression, and yet there can be justifications with seemingly little superfluity (as in the case of **EM3**) which causes complete distraction when trying to understand an entailment. Ultimately, what seems to be important is the location and shape of superfluity, but deciding upon what "shapes" of superfluity count as non-trivial needs to be investigated as part of future work.

One important point to consider, is that it might be possible to deal with the problems associated with superfluity by presentation techniques alone. It should be clear that the model does not pay any attention to how justifications are presented. For example, it is obvious that the ordering (and possibly the indentation) of axioms is important. It can make a big difference to the readability of justifications and how easy or difficult they are to understand, yet the model does not take into consideration how axioms will be ordered when a justification is presented to users. In the case of superfluity, it is conceivable that *strikeout* could be used to cross out the superfluous parts of axioms and this would dispel any problems associated with distracting superfluity. Figure 12.6 shows the helpful effect of strikeout on **EM3**. As can be seen, it immediately indicates that the problematic conjunct, $\exists$ prop1.C6, in the third axiom should be ignored. While strikeout seems promising, and has been observed to be very appealing to users, further experiments should be carried out to investigate the effects of superfluity and presentation techniques that might ameliorate the problems and issues surrounding superfluity and understanding.

EM3	**EM3**
C1 ⊑ C3	C1 ⊑ C3
C3 ⊑ C4	C3 ⊑ C4
C4 ≡ C5 ⊓ (∃ prop1.C6)	C4 ≡ C5~~ ⊓ (∃ prop1.C6~~)
C5 ≡ C7 ⊓ (∃ prop2.C8)	C5 ≡ C7~~ ⊓ (∃ prop2.C8~~)
C1 ⊑ ∃ prop1.C9	C1 ⊑ ∃ prop1.C9
C9 ⊑ C10	C9 ⊑ C10
C2 ≡ C7 ⊓ (∃ prop1.C10)	C2 ≡ C7 ⊓ (∃ prop1.C10)

Figure 12.6: **EM3** with and without strikeout

12.5 Discussion

This chapter has presented an investigation into the complexity of understanding justifications. A basic model has been presented which can be used to predict how easy or difficult it is for a person to understand a justification. Three key contributions of this work are: (1) It has shown that there are naturally occurring justifications that people with a range of backgrounds, including people who are very experienced in OWL, can find it difficult or impossible to understand; (2) It has shown that there are justifications that people can understand. This includes people with little or no training in OWL; and (3) It has presented a methodology and protocol that can be used to refine a justification complexity model. The main advantages of this experimental protocol is that minimal study facilitator intervention is required. This means that, over time, it should be possible to collect rich and varied data fairly cheaply and from geographically distributed participants. In addition to this, given a justification corpus and population of interest, the main experiment is easily repeatable with minimal resources and setup. Care must be taken in interpreting results and, in particular, the protocol is weak on "too hard" justifications as it cannot distinguish a model mislabeling from people failing for the wrong reason. However, this is arguably a feature of similar experiments, such as Newstead's [NBH^{+}06], and is a problem that is somewhat difficult to eliminate without using a more resource intensive protocol such as think-aloud.

Overall, while there is obviously more work to be done with regards to model refinement and evolution, as a first approximation the cognitive complexity model that was presented in this thesis fared reasonably well. In most cases, there was a significant difference in error proportion between model ranked easy and hard justifications. In the cases where error proportions revealed no difference better than chance alone, further small scale followup studies in the form of a more expensive talk-aloud study was used to gain an insight into the problems. These inspections highlighted an area for model improvement, namely in the area of superfluity. Unfortunately, without a deeper investigation, it is unclear how to rectify this in the model, whether the model needs rectifying at all, or whether presentation techniques alone can deal with the problems associated with superfluity. What is encouraging, is that superfluity was cited as causing problems with understanding. The experiments presented in this chapter have provided some preliminary indication that this is indeed a real problem in naturally occurring justifications, and that further investigation in this area is probably interesting and worthwhile.

Finally, it is worth noting that there has been a significant amount of work on predicting the complexity of understanding and the ease of maintainability of software. In particular, seminal work by McCabe [McC76], which devised a complexity metric known as *cyclomatic complexity* was based on the control flow paths through software. McCabe's work was followed by a plethora of other work, for example in [Hal77] Halstead uses various syntactic measures such as program vocabulary and program length to calculate *volume* and *difficult* of understanding of a program. In [Wey88] Weyuker provides a set of standard properties that software complexity measures should be attuned to. Some of the inspiration and ideas for the properties of the complexity model presented here were drawn from this work.

12.6 Conclusions

- There are naturally occurring justifications that are very difficult or impossible for a wide range of people to understand. This includes people who are very experienced with OWL.

- Justifications that contain non-obvious intermediate steps are difficult to understand. In particular, non-explicit synonyms of $\top$, trivial satisfaction of universal restrictions, and unfamiliar patterns of axioms all cause problems with justification understanding.

- A model which performs a syntactic inspection on the number of axiom and concept constructor types, in combination with spotting semantic phenomena such as synonyms of $\top$ and $\bot$, trivial satisfaction, can predict much better than chance alone whether or not a justifications is easy or difficult to understand.

- Error proportions can be used to test for the difficulty of justifications. They provide a reasonably accurate first pass at confirming model predictions, and do not require high levels of facilitator interaction with study participants. When anomalies arise, they can be followed up by more detailed think-aloud investigations.
- The issue of how superfluity affects understanding is non-trivial and needs to be investigated further.

Chapter 13

Justification Oriented Proofs

As evidenced by the results of the previous chapter, there are naturally occurring justifications that can be very difficult or impossible for people to understand. This includes people who have a significant amount of experience in OWL. Difficult to understand justifications typically contain non-obvious intermediate inference steps that need to be arrived at and brought together by the person reading the justification in order for them to understand how the justification supports the target entailment. In essence, the axioms in justifications are akin to the premises of a proof, and the entailment the conclusion, where it is left up to the reader of the justification to decide how to get from premises to conclusion. When people fail, or find it difficult, to spot *intermediate entailments, conclusions or steps* they can fail to understand why a justification supports the entailment in question, and hence fail to understand why the entailment holds in their ontology. This chapter proposes some ideas dealing with this issue. Specifically, it presents a conceptual framework for introducing helpful intermediate inference steps, called lemmas, into justifications. These lemmas, which are themselves explained by justifications, can be stitched together into a structure that this thesis calls a *justification oriented proof*. Part of the novelty of the framework is that complexity models, such as the one presented in the previous chapter, are used to choose these steps. While the work presented in this chapter is somewhat speculative in nature, it does provide a starting point and some leads for dealing with issues of justification understanding.

13.1 From Justifications Towards Proofs

The notion of "intermediate steps" that could guide a person through understanding a justification, raises the question of whether full blown proofs, such as *natural deduction* style proofs with inference rules, should be used for explaining entailments in OWL ontologies. One of the typical claims about natural deduction is that it mimics human reasoning—that is, it has a *strong cognitive adequacy* [Str92]. However, there is ongoing debate in the field of cognitive psychology about how human reasoning actually works. Some camps favour a "logic" or rule based account [Rip94], while others favour a "model" based account [JLB91]—even for simple cases of natural language based deduction, it is unclear which account is correct. It is therefore impossible to say whether or not natural deduction and similar proof systems mimic human reasoning. What is clear, is that representations that have a strong cognitive adequacy are not necessarily useable [Str92]. Hence, even if natural deduction has a strong cognitive adequacy, there is no guarantee that it is usable as a form of explanation for entailments in ontologies.

Ultimately, natural deduction style proofs are *not necessarily* the best form of explanation. Given the popularity of justifications in terms of tools support, the fact they are being widely and successfully used as a form of explanation (in stark contrast to any proof based explanation systems) indicates that a complete move to full blown proofs would be unwise. What is arguably needed, is something that lies between justifications and proofs. Given the popularity and conceptual simplicity of justifications, the work presented in this thesis uses them as building blocks for structures that begin to look like proofs, but are independent of any calculus or deduction rules. In essence, intermediate steps are introduced into a justification, which are themselves explained with justifications. This results in a directed acyclic proof graph of the form shown in Figure 13.1, and is called a *justification oriented proof.*

13.2 Justification Oriented Proofs

The main idea behind a justification oriented proof is depicted in Figure 13.1. The numbered lozenges represent axioms, with the leftmost lozenge, labelled η, representing the entailment of interest. The white lozenges labelled with "1" –

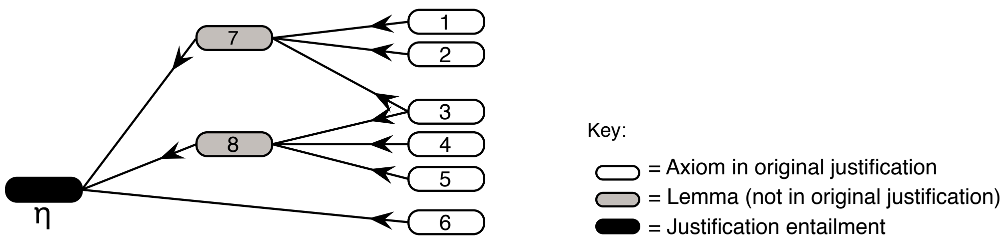

1
2
3
4
5
6
7
8
η
Key:
= Axiom in original justification
= Lemma (not in original justification)
= Justification entailment

Figure 13.1: A schematic of a Justification Oriented Proof

"6" represent exactly the axioms that appear in the original justification $\mathcal{J}$ for the entailment (and are therefore in the ontology as asserted axioms). Grey shaded lozenges represent *lemmas* that are entailed by $\mathcal{J}$ but are not in $\mathcal{J}$ as asserted axioms. For a given node, its direct predecessors constitute a justification for that node. This produces a weakly connected directed acyclic graph, with one sink node that represents the entailment of interest and a source node for each axiom in the justification. Hence, in the example shown in Figure 13.1, $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ is a justification for η with respect to the ontology that entails η. Axiom 7 is a lemma for axioms 1, 2 and 3 (conversely, axioms 1, 2 and 3 are a justification for axiom 7). Axiom 8 is a lemma for axioms 3, 4 and 5 (conversely axioms 3, 4 and 5 are a justification for axiom 8). Together axioms 6, 7 and 8 constitute a justification for η i.e. the entailment. Notice that axiom 3 participates in different justifications for different lemmas.

In essence, a justification oriented proof guides a person through the understanding of an original justification. Not only can lemmas make non-obvious intermediate steps explicit, they can also provide a chunking mechanism, which can help guide a user through a large and tedious to understand justification.

13.3 Related Work

The idea of using proofs as forms of explanation is obviously not new. Indeed, in some camps [BCR08, Kwo05], proofs are essentially regarded as *the* main form of explanation. However, the work that is presented in this paper is based on the intuitions mentioned in the introduction. That is, it is arguably more practical and more helpful to *not* show full blown proofs because (1) users already know and understand justifications, and (2) it avoids having to teach users a new calculus or deduction rules.

In [BFH00] Borgida uses a sequent calculus as the basis for explaining subsumption in $\mathcal{ALC}$. The proofs produced by Borgida's approach explicitly reference the inference rules that are used to go from one step to the next, and in this regard are fairly close to formal proofs and therefore not in the spirit of justification oriented proofs. Borgida briefly mentions the idea of sub-steps and weakenings as ways of deriving higher quality explanations.

Lingenfelder [Lin89] and Huang [Hua94] tackle the problem of presenting machine generated proofs to humans. In both cases, they attempt to address the problem that machine generated proofs are difficult for humans to understand. Lingenfelder remarks that even natural deduction proofs are at too low a level for human understanding, and that their length presents a difficulty in seeing "the important steps" and hinders understanding. Huang also argues that natural deduction proofs are also at too low a level, and develops *ND style proofs* that are at a higher level of abstraction. Interestingly, Lingenfelder sketches the idea of grouping proof steps together and applying lemmas. He also points out that it is necessary to distinguish between trivial steps and more complicated steps, possibly with use of a model.

In [dSSC$^+$08], da Silva et al. present proofs using the Inference Web tools. Trees are used to present proofs, where the nodes in tree (steps in the proofs) are determined using explicit low level inference rules. This is in contrast to justification/complexity oriented approach taken here.

Finally, in [Sch04] Schlobach introduces optimal interpolants, and so called *illustrations* that are intended to bridge the gap between subsumee and subsumer class expressions. The notion of lemmas and justifications oriented proofs as presented here are in the spirit of Schlobach's illustrations. However, the main difference is that Schlobach's work primarily deals with subsumption between two class expressions in isolation, whereas the work presented here deals with arbitrary entailments that arise from a set of axioms.

13.4 Proof Generation Framework

In what follows the framework for generating justification oriented proofs is presented. The framework consists of two main ideas: (1) The notion of justification lemmatisation, wherein subsets of a justification may be replaced with simple summarising axioms, which are known as *lemmas*. One justification is lemmatised into another justification. (2) The notion of stitching a series of lemmatised justifications into a justification oriented proof. First a definition of justification lemmatisation is presented and then a definition for justification oriented proofs is given.

13.4.1 Justification Lemmatisation

Given a justification $\mathcal{J}$ for an entailment η, the aim is to lemmatise $\mathcal{J}$ into $\mathcal{J}'$, so that $\mathcal{J}'$ *is less complex by some measure* and for *some purpose* than $\mathcal{J}$. With this notion in hand, lemmas for justifications can now be defined. First, an informal description is given, then a more precise definition is given in Definition 21.

Informally, a set of lemmas $\Lambda_{\mathcal{S}}$ for a justification $\mathcal{J}$ for η is a set of axioms that is entailed by $\mathcal{J}$ which can be used to replace some set $\mathcal{S} \subseteq \mathcal{J}$ to give a new justification $\mathcal{J}' = (\mathcal{J} \setminus \mathcal{S}) \cup \Lambda_{\mathcal{S}}$ for η. If, additionally, $\mathcal{J}'$ is less complex, by some measure, than $\mathcal{J}$. $\mathcal{J}'$ is called a *lemmatisation of* $\mathcal{J}$.

Various restrictions are placed on the generation of the set of lemmas $\Lambda_{\mathcal{S}}$ that can lemmatise a justification $\mathcal{J}$. These restrictions prevent "trivial" lemmatisations, an example of which will be given below. Before these restrictions are discussed, it is useful to introduce the notion of a *tidy* set of axioms. Intuitively, a set of axioms is *tidy* if it is consistent, contains no synonyms of $\bot$ (where a class name is a synonym of $\bot$ with respect to a set of axioms $\mathcal{S}$ if $\mathcal{S} \models A \sqsubseteq \bot$), and contains no synonyms of $\top$ (where a class name is a synonym of $\top$ with respect to a set of axioms $\mathcal{S}$ if $\mathcal{S} \models \top \sqsubseteq A$).

Definition 20 (Tidy sets of axioms). *A set of axioms $\mathcal{S}$ is* tidy *if $\mathcal{S} \not\models \top \sqsubseteq \bot$, $\mathcal{S} \not\models A \sqsubseteq \bot$ for all $A \in \mathsf{signature}(\mathcal{S})$, and $\mathcal{S} \not\models \top \sqsubseteq A$ for all $A \in \mathsf{signature}(\mathcal{S})$.*

The definition of lemmatisation that follows, mandates that a set of lemmas $\Lambda_{\mathcal{S}}$ must only be drawn from (i) the deductive closure of *tidy* subsets of the set $\mathcal{S} \subseteq \mathcal{J}$, (ii) from the *exact* set of synonyms of $\bot$ or $\top$ over $\mathcal{S}$.

Without the above restrictions on the axioms in $\Lambda_{\mathcal{S}}$, it would be possible to lemmatise a justification $\mathcal{J}$ to produce a justification $\mathcal{J}'$ that, in isolation, is simple to understand, but otherwise bears little or no resemblance to $\mathcal{J}$. For example, consider $\mathcal{J} = \{A \sqsubseteq \exists R.B,\ B \sqsubseteq E \sqcap \exists S.C,\ B \sqsubseteq D \sqcap \forall S.\neg C\}$ as a justification for $A \sqsubseteq \bot$. Suppose that *any* axioms entailed by $\mathcal{J}$, could be used as lemmas (i.e. there are no restrictions on the axioms that make up $\Lambda_{\mathcal{S}}$).

In this example, A is unsatisfiable in $\mathcal{J}$, meaning that it would be possible for $\mathcal{J}' = \{A \sqsubseteq E, A \sqsubseteq \neg E\}$ to be a lemmatisation of $\mathcal{J}$. Here, $\mathcal{J}'$ is arguably easier to understand than $\mathcal{J}$, but bears little resemblance to $\mathcal{J}$. In other words, $A \sqsubseteq E$ and $A \sqsubseteq \neg E$ are not helpful lemmas for $\mathcal{J} \models A \sqsubseteq \bot$. Similarly unhelpful results arise if lemmas are drawn from *inconsistent* sets of axioms, or sets of axioms that contain synonyms for $\top$.

Given the above intuitions and the notion of tidy sets of axioms, the notion of justification lemmatisation is defined as follows:

Definition 21 (Justification Lemmatisation). *Let $\mathcal{J}$ be a justification for η and $\mathcal{S}$ a set of axioms such that $\mathcal{S} \subseteq \mathcal{J}$. Let $\Theta_{\mathcal{S}}$ be the set of tidy subsets of $(\mathcal{S} \cup \delta(\mathcal{S}))$. Recall that $\mathcal{T}^\star$ is the deductive closure of a set of axioms $\mathcal{T}$. Let*

$$\Lambda_{\mathcal{S}} \subseteq \bigcup_{\mathcal{T} \in \Theta_{\mathcal{S}}} \mathcal{T}^\star \ \cup \ \{\alpha \,|\alpha \text{ is of the form } A \sqsubseteq \bot \text{ or } \top \sqsubseteq A, \\ \text{and } \exists \mathcal{K} \subseteq (\mathcal{S} \cup \delta(\mathcal{S})) \text{ that is consistent and } \mathcal{K} \models \alpha\}$$

$\Lambda_{\mathcal{S}}$ is a set of lemmas *for a justification $\mathcal{J}$ for η if, for $\mathcal{J}' = (\mathcal{J} \setminus \mathcal{S}) \cup \Lambda_{\mathcal{S}}$*

1. *$\mathcal{J}'$ is a justification for η over $\mathcal{J}^\star$, and,*
2. *Complexity$(\eta, \mathcal{J}')$ < Complexity$(\eta, \mathcal{J})$.*

The ability to lemmatise one justification into another justification *is a key process* in constructing a justification oriented proof. Given a regular justification $\mathcal{J}$ for η, $\mathcal{J}$ can be lemmatised into $\mathcal{J}_1$ for η. The axioms in $\mathcal{J}_1$ may then be inspected to determine which of them are lemmas – lemmas are axioms that are not in $\mathcal{J}$. Given a lemma $\alpha \in \mathcal{J}_1$ $(\alpha \notin \mathcal{J})$ a new justification $\mathcal{J}_2 \subseteq \mathcal{J}$ for α can be identified. If necessary, $\mathcal{J}_2$ can then be lemmatised into a simpler justification for α. Axioms in $\mathcal{J}_2$ can then be inspected and the process can be repeated as necessary. Ultimately the process builds up a justification oriented proof, the structure of which is defined below.

13.4.2 Justification Oriented Proofs

Definition 22 (Justification Oriented Proof). *A justification oriented proof for a justification $\mathcal{J}$ for an entailment η in $\mathcal{O}$ is a weakly connected directed acyclic graph $G = (V, E)$ such that $\mathcal{J} \subseteq V \subset \mathcal{J}^\star$ and either, $G = (\{\eta\}, \{\langle\eta, \eta\rangle\})$ or,*

1. *η is the one and only sink node in G,*

2. *$\mathcal{J}$ is the exact set of source nodes in G, and*

3. *For a given node, the set of predecessor nodes are a justification for the node over $\mathcal{J}^\star$.*

In summary, as shown in Figure 13.1, a node in a justification oriented proof that has incoming edges, is either a lemma or the entailment (sink node) itself. Source nodes (nodes with no predecessors) are the axioms in the original justification. Finally, given one justification $\mathcal{J}$ for η, there may be *multiple* justification oriented proofs, even if the set of lemmas in the proof is fixed.

Singleton Set Proofs Definition 22 admits justification oriented proofs for the case where a justification for an entailment is the singleton set containing the entailment itself. That is, given $\alpha \in \mathcal{O}$, it is the case that $\{\alpha\}$ is a justification for $\mathcal{O} \models \alpha$, and this has a corresponding justification oriented proof of $G = (V, E)$ where $V = \{\alpha\}$ and $E = \{\langle\alpha, \alpha\rangle\}$.

Proof Existence Given a justification $\mathcal{J}$ for $\mathcal{O} \models \eta$, there is always a justification oriented proof for $\mathcal{J}$ and η. This follows because either a Justification Oriented Proof takes the form of a singleton set proof as described above, or a Justification Oriented proof G can be trivially constructed by setting $G = (V, E)$ where $V = \mathcal{J} \cup \{\eta\}$ and $E = \{\langle\alpha_i, \eta\rangle \mid \alpha_i \in \mathcal{J}\}$.

Non-Edge-Uniqueness Given a justification $\mathcal{J}$ for $\mathcal{O} \models \eta$, then there can exist two justification oriented proofs $G = (V, E)$ and $G' = (V', E')$ such that $G \neq G'$, but $V = V'$ $(E \neq E')$.

A Comment about Proof Presentation It should be noted that, in the same way that raw unordered justifications are not presented directly to end users, it is unlikely the graph which constitutes a justification oriented proof should be presented *directly* to end users. Instead, the graph can be used as an input into some interactive presentation device.

13.5 The Use of Models to Select Proof Steps

As can be seen from Definition 21, justification lemmatisation depends upon the notion of justification complexity. More specifically, it depends upon whether one justification is more complex, by some measure and for some purpose, than another justification. In this framework, *complexity models* are used to assign complexity scores to justifications and determine whether one justification is more complex, than another. The framework makes *no commitment to a particular complexity model.* Indeed, models are intended to be pluggable. A model may depend upon the application in question and the intended audience. In the work presented here, the primary aim is to produce justification oriented proofs, which pick out difficult to spot lemmas, and chunk and summarise sets of heterogeneous axioms in justifications. With these goals in mind, the simple model presented in Chapter 12 is used. However, before continuing, models that deal with special use cases are first discussed. The main intention here, is to give a feel for how different models can be appropriate for different applications, and how different models may be plugged into the framework.

A Model for Deriving Proofs for Laconic Justifications

In Chapter 8 laconic justifications were presented as justifications whose axioms have no superfluous parts and whose parts are as weak as possible. Given $\mathcal{O} \models \eta$, a laconic justification oriented proof consists of a sink node η, and predecessors of η which are either (1) leaf nodes representing axioms contained in $\mathcal{O}$, or (2) are nodes representing axioms entailed by $\mathcal{O}$, for which each one has a predecessor representing an axiom contained in $\mathcal{O}$. Given a justification $\mathcal{J}$ for η, a simple complexity model for computing such proofs assigns a score of zero to $(\mathcal{J}', \eta)$ if $\mathcal{J}'$ is a laconic justification for η, a score of zero to $(\mathcal{J}', \alpha)$ if $\alpha \neq \eta$ and α is in the laconic justification in question, and $\mathcal{J}'$ is a singleton set containing an axiom from the original ontology, and otherwise, a score of one.

A Model for Deriving Proofs for Root/Derived Unsatisfiable Classes

Given an ontology $\mathcal{O}$ which contains unsatisfiable classes ($\mathcal{O} \models A \sqsubseteq \bot$ for some class name A in the signature of $\mathcal{O}$), a root unsatisfiable class [Kal06] is a class in the signature of $\mathcal{O}$ whose unsatisfiability does not depend on the unsatisfiability of any other class in the signature of $\mathcal{O}$. A derived unsatisfiable class is a class whose unsatisfiability depends on the unsatisfiability of some other class in the signature of $\mathcal{O}$. More precisely, given $\mathcal{O} \models A \sqsubseteq \bot$, A is a derived unsatisfiable class if there exists some class B such that $\mathcal{O} \models B \sqsubseteq \bot$ and there is a justification $\mathcal{J}_A \models A \sqsubseteq \bot$

and another justification $\mathcal{J}_B \models B \sqsubseteq \bot$ such that $\mathcal{J}_A \subsetneq \mathcal{J}_B$, otherwise, A is a root unsatisfiable class.

A suitable model that will lemmatise and "collapse" a subset that corresponds to a justification for a root unsatisfiable class (corresponding to $\mathcal{J}_B$ above) is as follows: Given $\mathcal{O} \models A \sqsubseteq \bot$, the model assigns a score of 1 to a justification $\mathcal{J}_A$ for $\mathcal{O} \models \eta$ if there exists a justification $\mathcal{J}' \subset \mathcal{J}$ for $\mathcal{J} \models B \sqsubseteq \bot$, where $\mathcal{J}' \neq \{B \sqsubseteq \bot\}$ and $\mathcal{J}'' = \mathcal{J} \setminus \mathcal{J}' \cup \{B \sqsubseteq \bot\}$ is a justification for $A \sqsubseteq \bot$ over the deductive closure of $\mathcal{O}$, the model otherwise assigns a score of 0.

13.6 An Algorithm for Generating Proofs

Given the above definitions, the main algorithms for generating proofs are presented below. There are three main algorithms: (1) GenerateProof, which takes a justification as an input and outputs a proof; (2) LemmatiseJustification, which takes a justification as an input and outputs either a lemmatised justification or the justification itself; (3) ComputeJPlus, which takes a justification and computes a set of axioms that are in the deductive closure of tidy subsets of the justification from which lemmas may be drawn. The GenerateProof algorithm uses the LemmatiseJustification as a sub-routine, and the LemmatiseJustification algorithm uses the ComputeJPlus algorithm as a sub-routine. Note that for the sake of brevity, the ComputeJPlus algorithm is not specified line by line in this thesis. Instead, a definition of $\mathcal{J}^+$ (Definition 23) is given below, and it is assumed that the algorithm simply computes $\mathcal{J}^+$ in accordance with this definition.

13.6.1 GenerateProof

The GenerateProof algorithm for computing justification oriented proofs is depicted in Figure 13.2. The basic idea is that, given an input of a justification $\mathcal{J}$ for η, a lemmatised justification $\mathcal{J}'$ for η is computed. $\mathcal{J}'$ is then used to initialise a justification oriented proof $\mathcal{P}$. For each node λ in the proof corresponding to an axiom in $\mathcal{J}'$, if λ is not in $\mathcal{J}$ then it is a lemma and a justification needs to be computed for it. In this case a new justification $\mathcal{J}''$ is computed for α' over $\mathcal{J}$. Next, $\mathcal{J}''$ is lemmatised to give $\mathcal{J}'''$ which is inserted into the proof $\mathcal{P}$. The process then repeats for lemmas in $\mathcal{P}$ that do not have any predecessors until none of the leaves in the proof are lemmas. Although not depicted in Figure 13.2, it is important to note that, in order to comply with Definition 22, there is a test in

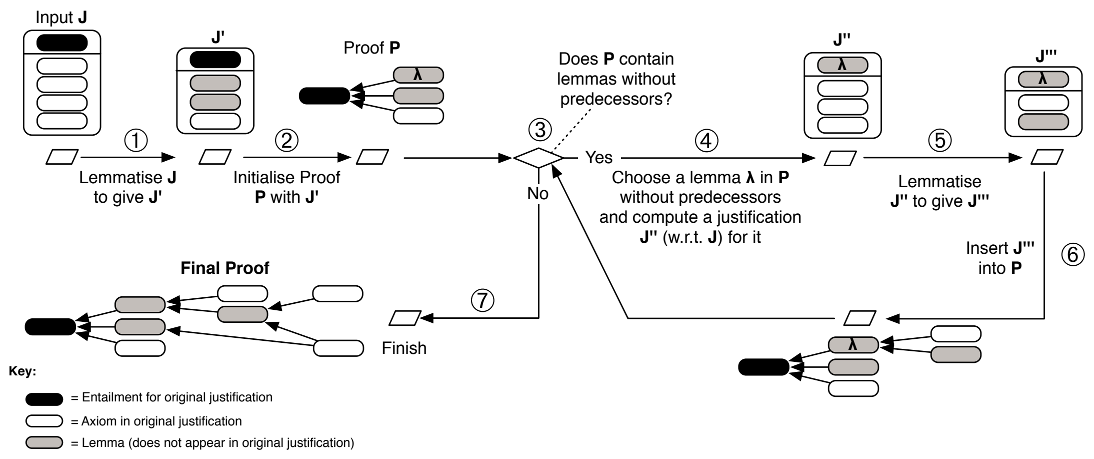

Input J
J'
Proof P
Does P contain lemmas without predecessors?
J''
J'''
①
Lemmatise J to give J'
②
Initialise Proof P with J'
③
Yes
No
④
Choose a lemma λ in P without predecessors and compute a justification J'' (w.r.t. J) for it
⑤
Lemmatise J'' to give J'''
⑥
Insert J''' into P
⑦
Finish
Final Proof
λ
Key:
= Entailment for original justification
= Axiom in original justification
= Lemma (does not appear in original justification)

Figure 13.2: An Algorithm for Generating Justification Oriented Proofs

step 6 to determine whether inserting $\mathcal{J}'''$ as a result of the lemmatisation process into $\mathcal{P}$ would result in a cyclic graph instead of a DAG. If this is the case, then an alternative lemmatisation of $\mathcal{J}''$ must be chosen (or if there are no alternatives then $\mathcal{J}''$ itself must be chosen) to insert into $\mathcal{P}$. This enforcement of non-cyclical proofs is also part of the mechanism that ensures the GenerateProof algorithm terminates. A discussion on termination is presented later.

13.6.2 LemmatiseJustification

The LemmatiseJustification algorithm is presented in Algorithm 13.1. The algorithm takes a justification J for η as its input and returns a justification J_{result} as its output. Either J_{result} is a lemmatisation of J or J_{result} is equal to J. In essence, the algorithm produces a lemmatised justification by computing a *filter* S_{tidy} on the deductive closure of tidy subsets of J, which obviously includes axioms that could lemmatise J. Justifications for η are then computed with respect to S_{tidy}. A complexity score is computed for each justification $J' \subseteq S_{tidy}$, which is compared to the complexity of J. If the difference between the score for J and the score for J' is positive then J' is selected as a lemmatisation of J. Algorithm 13.1 always terminates due to the fact that S_{tidy} is finite in size and hence there are a finite number of justifications for η with respect to S_{tidy}.

Algorithm 13.1 LemmatiseJustification(J, η)

```
1: if $J = \{\eta\}$ then
2:     return $\{\eta\}$
3: end if
4: $S_{tidy} \leftarrow$ ComputeJPlus$(J, \eta) \setminus \{\eta\}$
5: $X \leftarrow$ ComputeJustifications$(S_{tidy}, \eta)$
6: $c_1 \leftarrow$ ComputeComplexity$(J, \eta)$
7: $J_{result} \leftarrow J$
8: for $J' \in X$ do
9:     $c_2 \leftarrow$ ComputeComplexity$(J', \eta)$
10:    if $c_2 < c_1$ then
11:        $J_{result} \leftarrow J'$
12:    end if
13: end for
14: return $J_{result}$
```

13.6.3 ComputeJPlus

Definition 21 mandates that, for a justification $\mathcal{J}$, lemmas must be drawn from the deductive closure of tidy subsets of $\mathcal{J}$. However, the deductive closure of a set of axioms is *infinite*. For practical purposes it is necessary to work with a finite representative of the deductive closure that suffices for computing pleasing lemmatisations and pleasing justification oriented proofs. In addition to these practicalities, a finite representation of the deductive closure is needed because the ability to draw lemmas from an infinite set of axioms could lead to non-termination of the GenerateProof algorithm. In order to ensure termination, not only is it necessary to disallow cycles in the proof, but it is also necessary to introduce a filter on the deductive closure that produces a *finite* set of axioms, $\mathcal{J}^+$ from which lemmas may be drawn. In essence, $\mathcal{J}^+$ is some finite subset of the deductive closure of $\mathcal{J}$.

The question is, given a justification $\mathcal{J}$, what axioms should $\mathcal{J}^+$ contain? Although there is no definitive answer to this, it must be remembered that the ultimate goal is to include enough in $\mathcal{J}^+$ so that it is possible to produce a series of candidate lemmatised justifications, from which a “nice” one may be chosen using a complexity model. With this in mind, there are a number of possible options for $\mathcal{J}^+$ generation:

Generation with Sub-Concepts

One possibility is to specify $\mathcal{J}^+$ so that it contains axioms of the form $C \sqsubseteq D$[1] where C and D are built up from sub-concepts of axioms in $\mathcal{J}$. However, while such a strategy can go a long way to producing a set of axioms containing lemmas that could result in pleasing proofs, there could be axioms, which might be lemmas of choice, that are not be contained in the set. For example, given $\mathcal{O} = \{A \sqsubseteq \exists R.B, \exists R.B \sqsubseteq C, \mathsf{Trans}(R)\} \models A \sqsubseteq C$, a lemma of choice might be $\exists R.A \sqsubseteq \exists R.\exists R.B$ (entailed by $A \sqsubseteq \exists R.B$). However, with the above schema, based on sub-concepts, the class expression on the right hand side of the axiom ($\exists R.\exists R.B$) does not exists as a sub-concept in $\mathcal{J}$ and so the axiom would never be generated. What is needed is a set of class expressions that is rich enough so as to be able to build a rich set of axioms that constitute candidate lemmas. This is achieved using nested sub-concepts:

Generation with Nested Sub-Concepts

Definition 23 ($\mathcal{J}^+$)**.** *For a justification $\mathcal{J}$ for η, let $\mathcal{S}$ be the set of sub-concepts occurring in the axioms in $\mathcal{J} \cup \{\eta\}$ plus $\top$ and $\bot$. Let $\mathcal{S}'$ be the smallest set of class expressions such that $\mathcal{S}' \supseteq \mathcal{S}$ and $\mathcal{S}'$ contains class expressions of the form:*

- *$\neg C$ where $C \in \mathcal{S}'$ and C is not negated.*
- *$C_1 \sqcap \cdots \sqcap C_i$ or $C_1 \sqcup \cdots \sqcup C_i$ for $2 \leq i \leq |\mathcal{S}|$ and for any $C_j \in \{C_1, \ldots, C_i\}$ it is the case that $C_j \in \mathcal{S}$ or $C_j = \neg C$ for some $C \in \mathcal{S}$ where C is not negated.*

Now, let $d = |\mathcal{J}| \times c$ where c is the maximum modal depth [BCM+03] of the class expressions in $\mathcal{S}$. Let R be a property in the signature of $\mathcal{J}$ and m be the sum of all numbers occurring in cardinality restrictions. Let $\mathcal{S}''$ be the smallest set of class expressions such that $\mathcal{S}'' \supseteq \mathcal{S}'$ and $\mathcal{S}''$ contains class expressions of the form:

- *$\exists R.C$, $\forall R.C$, $\geq nR.C$ or $\leq nR.C$, where $C \in \mathcal{S}''$, the modal depth of C is no greater than d, and $n \leq m$.*
- *$\exists R.\{a\}$, where a and R are in the signature of $\mathcal{J}$ or η.*

[1]For OWL syntactic variations of these axioms could be constructed, for example $\mathsf{domain}(R, C)$ instead of $\exists R.\top \sqsubseteq C$

- *$\neg C$ where $C \in \mathcal{S}''$ and C is not negated.*

Given $\mathcal{S}''$, $\mathcal{J}^+$ is now defined as the set of axioms of the form $C \sqsubseteq D$, where C and D are substituted for class expressions in $\mathcal{S}''$, R and S are substituted for property expressions in $\mathcal{J}$, a and b are substituted for individuals in the signature of $\mathcal{J}$, and for each axiom $\alpha \in \mathcal{J}^+$, there exists a tidy subset $\mathcal{J}' \subseteq \mathcal{J}$ such that $\mathcal{J}' \models \alpha$.

The ComputeJPlus algorithm in now defined to compute $\mathcal{J}^+$ in accordance with Definition 23. Since $\mathcal{S}$ is finite, $\mathcal{S}''$ is also finite and therefore $\mathcal{J}^+$ is also finite. Therefore, there are finite number of justifications for an entailment η with respect to $\mathcal{J}^+$, hence GenerateProof algorithm is guaranteed to terminate.

13.7 The Feasibility of Computing Justification Oriented Proofs

In order to get a feel for the practicalities of computing justification oriented proofs, the GenerateProof algorithm and its sub-routines, and the complexity model shown in Table 12.1 were implemented in Java using the OWL API. The algorithm has two basic optimisations. First, $\mathcal{J}^+$ is computed incrementally and the number of entailment checks is minimised in the obvious way, for example, given the arbitrary concepts C, D and E, if it is found that $\mathcal{J} \not\models C \sqsubseteq D$ then an entailment test is not also performed on $\mathcal{J}$ for $C \sqsubseteq D \sqcap E$. Second, justifications in the LemmatiseJustification algorithm are computed one by one rather than all at once. This means that if a justification $\mathcal{J}'$ is found as a lemmatisation of $\mathcal{J}$ this justification is selected rather than continuing to look for one of lower complexity. If necessary, $\mathcal{J}'$ could be lemmatised to produce a justification of possibly lower complexity.

The implemented algorithm, with the Pellet reasoner, was tested against the ontologies listed in Table 13.1. For each ontology, a maximum of 5 justifications per entailment of the form $A \sqsubseteq B$, $A \sqsubseteq \bot$ and $A(a)$ were computed. Proofs were then computed for these justifications. Times for computing the justifications, and times for computing proofs were measured and averaged.

Generally speaking, if it is possible to compute a justification for an entailment, it appears possible to compute a justification oriented proof for that justification and entailment. In all cases, the time required to compute the proof is

Table 13.1: Mean Times for Computing Justifications and Proofs

Ontology Expressivity/Axioms	**Just. Size (Mean/SD/Max)**	**Just. Time (mean ms)**	**Proof Time (mean ms)**
Generations ($\mathcal{ALCOIF}$/38)	4 / 2.1 / 8	31	2034
Economy ($\mathcal{ALCH}$/1625)	2 / 0.6 / 6	32	144
People+Pets ($\mathcal{ALCHOIN}$/108)	4 / 2.5 / 16	31	801
Tambis ($\mathcal{SHIN}$/595)	8 / 4.1 / 21	1047	244987
Nautilus ($\mathcal{ALCF}$/38)	3 / 2.0 / 6	20	758
Transport ($\mathcal{ALCH}$/1157)	5 / 2.1 / 9	19	469
University ($\mathcal{SOIN}$/52)	5 / 2.1 / 9	21	1738
PeriodicTable ($\mathcal{ALU}$/100)	4 / 9.9 / 36	72	1026
Chemical ($\mathcal{ALCHF}$/114)	8 / 1.2 / 11	38	3690

at least an order of magnitude higher than the time required to compute a justification. The difference is particularly striking for the Tambis ontology, where there were several justifications for which it took a significant time to perform entailment checking while computing $\mathcal{J}^+$ and then compute justifications over $\mathcal{J}^+$. The implementation, although naive, with plenty of room for further optimisation, indicates that it ought to be practical to compute proofs for entailments in real ontologies, and that the optimisation of the algorithms and other aspects of justification oriented proofs merit further investigation.

13.8 Examples

This section presents two examples of justification oriented proofs that get constructed using the above framework with the complexity model from Chapter 12 plugged into it. The examples illustrate the kinds of lemmas that get introduced into proofs and illustrate what is possible using the complexity model. Figure 13.3 shows a justification oriented proof for the justification shown in Figure 12.1, and Figure 13.4 shows (part of) a justification for the justification shown in Figure 12.2.

In the tree style of presentation used, the children of an axiom represent a justification for that axiom. Original axioms that appear in a justification are shown in a bold font and lemmas shown in a lightweight font. It should be

Entailment : $\mathsf{Person} \sqsubseteq \bot$

- $\mathbf{Person} \sqsubseteq \neg\mathbf{Movie}$
- $\top \sqsubseteq \mathsf{Movie}$
    - $\forall\mathsf{hasViolenceLevel}.\bot \sqsubseteq \mathsf{Movie}$
        - $\forall\mathsf{hasViolenceLevel}.\bot \sqsubseteq \mathsf{RRated}$
            - $\mathbf{RRated} \equiv (\exists\mathbf{hasScript}.\mathbf{ThrillerScript}) \sqcup (\forall\mathbf{hasViolenceLevel}.\mathbf{High})$
        - $\mathsf{RRated} \sqsubseteq \mathsf{Movie}$
            - $\mathbf{RRated} \sqsubseteq \mathbf{CatMovie}$
            - $\mathbf{CatMovie} \sqsubseteq \mathbf{Movie}$
    - $\exists\mathsf{hasViolenceLevel}.\top \sqsubseteq \mathsf{Movie}$
        - $\mathbf{domain}(\mathbf{hasViolenceLevel}, \mathbf{Movie})$

Figure 13.3: A schematic of a justification oriented proof for the justification shown in Figure 12.1

noted that the presentation style used for the examples is merely for illustrative purposes.

As an example of how the read the justification oriented proofs consider the proof shown in Figure 13.3, which is for the entailment $\mathsf{Person} \sqsubseteq \bot$. This is entailed by the justification $\{\top \sqsubseteq \mathsf{Movie}, \mathbf{Person} \sqsubseteq \neg\mathbf{Movie}\}$, which is presented at the root level. In the justification oriented proof the axiom $\top \sqsubseteq \mathsf{Movie}$ is not shown in bold, which means that it is a lemma. Its justification is

$$\{\forall\mathsf{hasViolenceLevel}.\bot \sqsubseteq \mathsf{Movie}, \exists\mathsf{hasViolenceLevel}.\top \sqsubseteq \mathsf{Movie}\}$$

which corresponds to its child nodes in the justification oriented proof tree. The rest of the lemmas and axioms can be read in a similar way. For example, RRated is a Movie because the ontology contains two axioms: $\mathsf{RRated} \sqsubseteq \mathsf{CatMovie}$ and $\mathsf{CatMovie} \sqsubseteq \mathsf{Movie}$. For more examples, a selection of videos of justification oriented proofs may be found online at:

`http://www.cs.man.ac.uk/~horridgm/justificationorientedproofs`

13.9 Discussion

The previous chapter presented evidence which shows that people who are experienced with OWL can find justifications difficult or impossible to understand. This

Entailment : Newspaper(DailyMirror)

- **reads**(**Mick DailyMirror**)
- **Tabloid** ⊑ **Newspaper**
- ∀reads.Tabloid(Mick)
  - **WhiteVanMan** ⊑ ∀**reads**.**Tabloid**
  - WhiteVanMan(Mick)
    - Man ⊓ ∃drives.(Van ⊓ WhiteThing) ⊑ WhiteVanMan
      - **WhiteVanMan** ≡ **Man** ⊓ ∃**drives**.(**Van** ⊓ **WhiteThing**)
    - **WhiteThing**(**Q123ABC**)
    - **drives**(**Mick Q123ABC**)
    - Man(Mick)
      - Adult ⊓ Male ⊓ Person ⊑ Man
        - **Man** ≡ **Adult** ⊓ **Male** ⊓ **Person**
      - Adult(Mick)
        - ...
      - **Male**(**Mick**)
      - Person(Mick)
        - hasPet(Mick Rex)
          - ...
        - **domain**(**hasPet Person**)

Figure 13.4: A schematic of a justification oriented proof for the justification shown in Figure 12.2

chapter has presented a possible solution to this problem in the form justification oriented proofs.

The key aspect of justification oriented proofs is that the steps in them are only governed by the entailment relation and the notion of a justification rather than a fixed set of inference rules. This makes justification oriented proofs rather general. It means they are not specific to a given logic and they are not specific to a particular proof calculus. The benefits of this are that the notion applies to other (monotonic) logics, and from a usability point of view users of tools such as Protégé-4, who could benefit from justification oriented proofs, do not have to learn the principles and ideas behind a proof calculus as they only have to know the semantics of the language (OWL).

When generating justification oriented proofs a balance must be struck between making individual steps easy to understand, or work through, and making the proof too fine-grained. In this respect justification oriented proofs could suffer from granularity problems in the same way that any other proof system could. When proofs become too fine-grained it is easy to imagine there is a danger that a person trying to understand a proof can lose sight of how the axioms in the ontology interplay with each other and cause the entailment to hold—while the person can verify that each step of the proof is correct, they lose sight of the bigger picture.

Chapter 14

Conclusions

This chapter summarises the significance of the contributions made by this thesis. Some outstanding issues are discussed and suggestions for future work are presented.

14.1 Thesis Overview

This thesis has focused on advancing state of the art knowledge and techniques in the area of justification based explanation in ontologies. Justifications are widely used in many tools and are the dominant form of explanation in the world of OWL. They are also essential to many applications and services which require them for purposes other than explaining entailments. Despite this, there were open questions and issues with justification based explanation. Specifically in the area of computing justifications, fine-grained justifications and understanding justifications. This thesis set out to answer and investigate some of these questions and issues.

14.2 Summary of Contributions

Broadly speaking, the major contributions of this thesis are therefore:

1. A thorough evaluation of justification finding algorithms, which has provided strong evidence that shows how these algorithms perform in practice, and has shed new light on the justificatory structure of real world ontologies.

2. A significant advancement in the area of fine-grained justifications, with the introduction of Laconic and Precise Justifications. This includes the design, optimisation, implementation of algorithms for computing Laconic Justifications, and a thorough evaluation of these algorithms.
3. User studies that provide an insight into the way that people understand justifications. The development of a model for predicting the complexity of understanding of justifications, and a framework for the lemmatisation of justifications into justification oriented proofs.

On a practical level, the implementations of justification finding algorithms, both for regular and laconic justifications, are robust enough for use in ontology development environments today and other applications where justification finding is used as an auxiliary service. It is likely they will prove to be of real benefit to the Description Logic and OWL communities.

14.3 Summary and Significance of Main Results

The main results of this thesis are summarised below by topic.

14.3.1 Evaluation of Justification Finding Algorithms

Before this thesis, there was a lack of a large convincing body of evidence showing how robust and highly optimised justification finding algorithms perform in practice on naturally occurring ontologies. Such evidence is important because many applications and areas of research beyond justification finding rely on the good performance of these algorithms. Showing that the algorithms are robust and perform well in practice provides a sturdy and reassuring platform for these applications and future research. It also helps to settle an ongoing debate in the community which centres around the misunderstanding and misinterpretation of research results, bad experience with prototypical implementations in tools, and doubts about whether these algorithms are usable in practice. Chapters 3–6 of the thesis therefore dealt with the topic of evaluating justification finding algorithms.

The experiments carried out in Chapter 6 are far more thorough, systematic and exhaustive than any other justification finding experiments that have been reported in the literature to date. The ontology corpus that was used contains a wide selection of ontologies that vary greatly in expressivity, modelling style and size. The collection is defined by a third party and none of them were cherry picked to show good performance or because they contained interesting features. Moreover they are all naturally occurring, application oriented ontologies. The main results are:

- It is practical to compute all justifications for entailments in naturally occurring ontologies. This includes entailments that have hundreds of justifications. The rate of failure to find all justifications for entailments is very low, and the algorithms perform well in practice and scale well given the large numbers of justifications and the size of these justifications. The main reason for the scalability is due to the justificatory structure of naturally occurring ontologies, in particular, the fact that real justifications tend to overlap.

- The limiting factor for computing all justifications is hitting set tree size. Failures that do occur are largely due to timeouts during the hitting set tree construction process, and arise due to the fact that there are naturally occurring ontologies that contain entailments with hundreds, or even thousands, of justifications. Large numbers of justifications like these result in hitting set trees with millions of nodes. The kinds of failures that occurred in the experiments, where there were large numbers of large justifications, suggest that this is a hard problem to solve, and that the bounds of tractability come into play. However, future research should focus on trying to manage this and looking at ways to partition the problem.

- The performance of optimised black-box algorithms compares favourably with the performance of glass-box algorithms for finding a single justification and finding all justifications for an entailment. For consistent ontologies black-box and glass-box times are typically of the same order of magnitude. This result, coupled with the fact that it is highly non-trivial to implement or augment an existing reasoner with glass-box tracing, suggests that it is not worthwhile implementing tracing solely for the purpose of generating justifications for explaining entailments. Indeed, the best use of reasoner development resources, as far as justification finding is concerned, would be to expend effort in ensuring that entailment checking is as optimised and as robust as it can be.

- Entailments in naturally occurring ontologies can have extremely large numbers of justifications that are well into the hundreds. This is an interesting finding that does not appear to have been previously documented. It is important because it has implications for both justification finding systems and end user facing systems which have to deal with and present these large numbers of justifications.

- In contrast to consistent ontologies, it appears that timeouts are more likely to occur when computing justifications for inconsistent ontologies. There are two reasons for this: (1) Some of the optimisations that are available for use with consistent ontologies, such as modularisation, cannot be used with inconsistent ontologies, and (2) The justifications for entailments in consistent ontologies are typically numerous. While the corpus of inconsistent ontologies used in the experiments is not particularly large, the ontologies in it are from disparate sources and are rather diverse, which provides a level of confidence that this is a common phenomenon.

Although the BioPortal ontologies are confined to modelling bio-medical knowledge, the expressivity of them shows that many bio-medical ontologies need highly expressive Description Logics for modelling purposes. Indeed, the BioPortal corpus is not simply confined to some level of lightweight expressivity that is only typical of bio-medical ontologies. More over the wide ranges of sizes, number of non-trivial entailments, number of justifications per entailment, and sizes of justifications provide a level of confidence that the results obtained in empirical investigation can be generalised to ontologies that are outside of the BioPortal corpus.

14.3.2 Laconic and Precise Justifications

The major contribution of this thesis has been the work on Laconic and Precise Justifications; the definitions of them, algorithms for computing them and an extensive evaluation of these algorithms on a large corpus of realistic ontologies. Prior to this work, fine-grained or precise justifications had been identified as being important in the literature. The superfluity that can exist in regular justifications had been slated as being distracting, causing problems with understanding, and causing problems with repair. Despite this, and the fact that there has been some prior work on fine-grained justifications, there was no proper definition for them. All previous approaches use intuition to characterise them, or rely upon "definition by implementation". This was troublesome because it made it difficult to extend the notion of fine-grained justifications to more expressive Description Logics, it made it difficult to design and show the correctness of algorithms for computing fine-grained justifications, and it made it difficult to properly analyse their properties and the phenomena associated with superfluity and only paying attention to parts of axioms. Providing proper definitions for fine-grained justifications, characterising and defining superfluity and, pinning down what parts of axioms are, solves these problems and opens up the field for deeper investigation. Chapters 7–11 therefore focused on fine-grained justifications. The main results are as follows:

- A proper definition has been provided for *Laconic and Precise Justifications*. This is the first proper definition of fine-grained justifications to be published. Although presented in the context of $\mathcal{SHOIQ}$, at a conceptual level the definition is essentially independent of the description logic used. In particular, it centres around the the crisp notion of $\top\bot$-superfluity and the weakening of parts of axioms, both of which are defined using the standard notion of subconcepts in combination with Plaisted and Greenbaums's structural transformation. The other major advantage of this generality is that the definition is not tied to a particular implementation technique. Based on the definition, it is possible to design algorithms for detecting and computing laconic justifications and check that they produce correct results, rather than the result being correct because they are what the algorithm produces.

- A proper characterisation and definition of justification masking has been provided. This thesis has identified and defined four types of masking: Internal masking, External masking, Cross-masking, and Shared-core masking and has provided proper definitions for them. This is a significant step forward, as all previous descriptions of masking were based on intuitions and "definition by example". This is despite the fact masking was identified in the literature as one of the primary motivations for fine-grained justifications. Moreover, as the empirical investigation in Chapter 11 shows, masking is prevalent in naturally occurring ontologies and is therefore a real and important phenomenon. Out of the 72 BioPortal ontologies, 53 exhibited some kind of masking (9 internal, 23 external, and 53 shared-core). In some cases, extreme levels of masking were observed, with some entailments having around 14-18 regular justifications but around 400-500 laconic justifications.

- Syntax is a crucial aspect in the definition of laconic and precise justifications, and in the definition masking. Indeed, entailment-based notions alone are not powerful enough to decide whether or not a justification is laconic or whether or not masking is present. The interplay between syntax and semantics was the most challenging aspect of coming up with a definition. At times, trying to pay homage to semantics and notions of weakness seemed completely at odds with respecting the syntax, and to some extent the usability, of laconic justifications. Ignoring syntax is not possible, because all reasonable intuitions of laconic justifications break down, and ignoring semantics is not possible because it prevents phenomena such as the notion of justification masking due to weakening being captured.

- It is possible to design and optimise algorithms for computing laconic justifications that form well in practice. Although some care was necessary when considering how to compute preferred laconic justifications using optimised algorithms, the definitions of laconic and precise justifications are such that the design and implementation of algorithms for computing them is a fairly straightforward process. It is not necessary, for example, to modify tableau reasoner internals, and implementations can be based on existing off-the-shelf justification finding services.

- The empirical evaluation shows that it is practical to compute laconic justifications for entailments in naturally occurring ontologies. Moreover, runtime performance is perfectly acceptable for use in ontology debugging environments and for use in ancillary services which provide support to other explanation services.

14.3.3 Understanding Justifications

Over the years, there has been anecdotal evidence from the OWL community which has expressed the view that some naturally occurring justifications are difficult or impossible to understand. Prior to this thesis, it was not clear whether this was true, and if so, what degree of action was necessary to ameliorate the problem. Hence, it seemed fruitful to investigate how people deal with justifications, what makes certain justifications difficult to understand, and suggest a framework that could be used for coping with any problems when they arise. Chapters 12 and 13 therefore concentrated on the understanding of justifications. The main results are:

- It was found that a range of people, from neophytes to those with several years of experience with OWL, can find some naturally occurring justifications very difficult or impossible to understand. Multiple types of axioms and concept expressions, specific phenomena such as non-explicit synonyms of $\top$, trivial satisfaction, and unfamiliar patterns of axioms in subsets of a justification cause problems with understanding. What is important are the steps that must be taken to get from the axioms in a justification to the target entailment, and naturally occurring justifications can contain extremely difficult steps where the line of reasoning is highly non-obvious. These results were captured in a complexity model for predicting how easy or difficult a justification is to understand.

- This thesis has contributed a workable protocol for the validation and evolution of a complexity model. The protocol is a user study based approach where error proportions are used to assess the actual difficulty of justifications, in combination with followup think-aloud studies, which are smaller and more focused, and can be used to investigate specific anomalies. The advantage of this approach is that such studies can be carried out remotely without the need for constant facilitator supervision. The use of this protocol showed that, although not perfect, the complexity model does have predictive power and performs *significantly* better than chance alone.

- Superfluity was observed to cause problems with understanding. A series of eye-tracking experiments revealed that, in some circumstances, people can fixate on superfluous parts in justifications, and that this can cause difficulties in understanding. However, some justification superfluity can be completely trivial to the point where it is irrelevant for understanding. The reasons as to what makes certain kinds of superfluity distracting and harmful, and other kinds of superfluity inconsequential for the purposes of understanding are unclear. What is clear, is that this is a non-trivial problem, and blindly throwing a superfluity component into the model would easily reduce the quality and predictive power of the model. The effect of superfluity on understanding should be investigated as part of future research.

- Justification oriented proofs were introduced as a speculative solution and coping aid for situations where people cannot understand justifications. They centre around the idea of using automated lemma generation techniques to introduce helpful intermediate inference steps into a justification. In essence, justifications get broken down into smaller chunks which guide people from the axioms in the original justification through to the target entailment. They are somewhat different from the natural deduction style proofs that have been proposed as forms explanation elsewhere in the literature. In particular, the steps contained in them are not based on a fixed set of inference rules. Instead, they are based solely on justifications and the entailment relation, with the size and frequency of steps being chosen by some complexity model. The hope is that a model based approach will ensure that proofs do not become too fine-grained, so as to avoid users switching into a proof checking mode and losing sight of why the entailment holds in an ontology.

14.4 Outstanding Issues

This thesis set out to address several issues with justification based explanation. While it has undoubtedly “advanced the cause”, there are some outstanding issues which ought to be addressed:

14.4.1 Dealing with Redundancy in Justifications

As discussed in Section 8.8 on page 153, the definition of laconic justification only deals with superfluity and does not deal with redundancy in the sense defined by Quine in [Qui52]. This was a deliberate design decision that was taken to keep the definition of laconic justifications simple, and more importantly make it practical to implement a weakening based algorithm for computing them. An investigation into redundancy and its prevalence in real ontologies was outside the scope of this thesis. However, it should be investigated and its relationship to laconic justifications should be determined.

14.4.2 Presentation of Laconic Justifications

Work needs to be done to establish optimal presentation and interaction mechanisms for laconic justifications. The main challenge is to come up with smooth ways in which to relate axioms in laconic justifications back to the asserted axioms from which they were derived. For simple occurrences of superfluity, the strikeout mechanism used in Swoop, shown in Figure 7.5 on Page 7.5, is intuitive to use and particularly effective. However, it is not clear how well this mechanism would extend to the weakening of cardinality restrictions, or the weakening of axioms when only one direction of an implication is required. The presentation of the various masking phenomena poses further interesting challenges, in particular the indication of internal masking and bringing together shared cores.

14.4.3 Further Optimisation of Algorithms for Computing Laconic Justifications

Out of the two algorithms for computing laconic justifications, the incremental π-based algorithm proved to be the most robust. However, there was still a small percentage of entailments in the NCI ontology, to name one, for which it was not possible to compute any preferred laconic justifications. The reason for this was due to extremely deep concept nesting in a handful of (less than 5) axioms. It could be possible that a hybrid algorithm which involves partial use of the δ-transformation, (structural transformation) in combination with the π based transformation would ameliorate the situation. The π could be applied to all axioms whose subconcept nesting is below some threshold (that would need to be determined by analytical and empirical analyses), and a variant of the δ-transformation could be applied to axioms containing subconcept nesting above this threshold. This would hopefully avoid the blow up in the number of justifications when using the δ-transformation due to cross-masking, and would avoid the blow up in size of $\pi(\mathcal{O})$ due to axioms with deep subconcept nesting.

14.4.4 Iteration and Refinement of the Complexity Model

The complexity model presented in Chapter 12 should be consider as a reasonably advanced starting point for further model refinement and validation. More experimental cycles on a broader test set of justifications with more participants should obviously be carried out. The main problem here is that to achieve a high level of confidence that the model performs well on a particularly large corpus, such as BioPortal, it would be necessary to sample something in the region of 1000 justifications. This is obviously prohibitively expensive. Ultimately, some compromise has to be reached here, and in any case, it is arguable that only enough evidence has to be amassed to really convince people that the model works well in practice.

Differences in results from Experiment 8 and Experiment 9 indicate a difference in performance between participants with moderate experience in OWL (PhD Students, Research Associates etc.), and people with little experience in OWL (MSc students who had only just learnt OWL). It would be interesting to investigate these differences, and establish how beginners approach understanding justifications compared with people who are experienced in OWL. Not only could this information be used to improve the complexity model, it could be informative for people who design tutorials and training courses.

14.4.5 Evaluation of Justification Oriented Proofs

Justification oriented proofs were introduced as a speculative solution to the problem of justification understanding. A thorough evaluation of justification oriented proofs is therefore beyond the scope of this thesis. However, it is obvious that there is far more work to be done here. In particular, there is no guarantee that current approach is completely immune to producing fine-grained proofs. It would therefore be worth investigating granularity problems, determining to what extent justification oriented proofs are susceptible to them, and devising some way of striking the right balance.

Another large piece of work that needs to be done is the design of a smooth presentation and interaction mechanism for use in tools like Protégé-4. This is certainly a non-trivial task. In the presentation of proofs in this thesis, a top-down summarising approach from entailment to premises was used for illustrative purposes. However, it is likely that a bottom-up approach from premises to entailment would fare as well if not better. Finally, only when an interaction mechanism is in place can justification oriented proofs be evaluated from a human perspective. Studies need to be carried out to determine the efficacy of justification oriented proofs. For completeness justification oriented proofs should be compared with regular justifications, laconic justifications, and classic natural deduction style proofs. While early indications from demonstrating tools at conferences and workshops suggest that justification oriented proofs would be popular with users, a formal user study is of course needed.

14.5 Future Work

The process of carrying out experiments on the BioPortal and the results obtained from these experiments, in particular the picture gained about the justificatory structure of realistic ontologies, suggests some avenues for further work which are presented below.

14.5.1 Dealing with Multiple Justifications

The results presented in Chapters 6 and 10 show that there can be large numbers of naturally occurring regular justifications and preferred laconic justifications per entailment. For example, several ontologies contained entailments with hundreds of justifications. While huge numbers of justifications can be computed, presenting them to people for assimilation becomes a significant problem. At a basic level, strategies for ranking and ordering justifications should be pursued so that justifications salient for repair or understanding can be presented first. Obviously, metrics for ranking must be developed for this to work.

When justifications for an entailment exhibit a high degree of overlap it might be worthwhile in only presenting the difference between them. A more advanced strategy would be to find common lemmata which summarise these differences.

For example, the ontology

$$
\begin{aligned}
\mathcal{O} = \{A &\sqsubseteq B && (\alpha_1)\\
B &\sqsubseteq C && (\alpha_2)\\
A &\sqsubseteq \exists R.D && (\alpha_3)\\
\exists R.\top &\sqsubseteq B && (\alpha_4)\\
\top &\sqsubseteq \forall S.B && (\alpha_5)\\
R &\equiv S^{-}\} \models A \sqsubseteq C && (\alpha_6)
\end{aligned}
$$

contains three justifications for $A \sqsubseteq C$: $\mathcal{J}_1 = \{\alpha_1, \alpha_2\}$, and $\mathcal{J}_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, and $\mathcal{J}_2 = \{\alpha_2, \alpha_3, \alpha_5, \alpha_6\}$. All of these justifications entail $A \sqsubseteq B$, and this could be used as a lemma to highlight the both the commonalities and differences amongst them. That is, $\mathcal{J}_{base} = \{A \sqsubseteq B,\ B \sqsubseteq C\}$ should be considered a *base justification* with the lemma $A \sqsubseteq B$ being justified by fragments of the other justifications, for example $\mathcal{J}_{A \sqsubseteq B} = \{A \sqsubseteq \exists R.D,\ \exists R.\top \sqsubseteq B\}$. The challenge of course is finding helpful common lemmata, which may be axioms with complex class expressions.

14.5.2 Justifications for Ontology Comprehension

As seen in the results of the empirical evaluation, there are a significant number of BioPortal ontologies that are logically rich. While some impression of the richness may be gleaned from looking at the DL expressivity metrics presented in Table 5.1 on page 93, these metrics are rather coarse-grained and opaque. That is, DL expressivity does not give an indication of the part played by axioms in the ontology in entailing structures such as the concept hierarchy. The use of the notion of non-trivial entailments, counting the number of justifications per entailment, their average size, their overlap and their expressivity in relation to the expressivity of the ontology provides a deeper picture of what the ontology is like from a logic-based perspective.

The kind of logic-based metrics that are useful and the phenomena that they illustrate seems worthy of further investigation. As well as providing a deeper picture of the contents of a single ontology it could be used to facilitate the comparisons of multiple ontologies as well as drawing out commonalities and differences between different collections of ontologies.

14.5.3 Reasoner Benchmarking

Computing justifications stresses parts of reasoners that do not necessarily get a thorough work out during classification. This is due to the fact that justification finding, and black-box justification finding in particular, typically requires large numbers of entailment checks on many different sets of axioms. Since many reasoner benchmarks concentrate on classification, using justification finding performance as a metric could provide a new interesting dimension for comparing reasoner performance.

14.5.4 Comparison of Different Corpora

Much of the corpus used for the empirical evaluation of algorithms presented in this thesis has been based on ontologies found in the BioPortal. At the time of writing, third party non-biomedical-ontology installations of the BioPortal software are coming on line. An interesting investigation would be to compare the repositories of ontologies from different communities, in terms of non-trivial entailments, justifications per entailment etc. and see how the justificatory structure and modelling style varies from one community to another.

Bibliography

[ABB$^+$00] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Alan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasaerskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

[AGM85] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[Baa91] Franz Baader. Augmenting concept languages by transitive closure of roles: an alternative to terminological cycles. In *IJCAI'91: Proceedings of the 12th international joint conference on Artificial intelligence*, pages 446–451, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In eslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 364–369. Professional Book Center, August 2005.

[BBMR89] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: a structural data model for objects. In James Clifford, Bruce Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 58–67. Association for Computing Machinery, Inc. (ACM), June 1989.

[BCH06] Claudia Bartz, Amy Coenen, and Woi-Hyun Hong. Participation in the International Classification for Nursing Practice (ICNP). In Arie Hasman, Reinold Haux, Johan van der Lei, Etienne De Clercq, and Francis H. Roger France, editors, *Ubiquity: Technologies for Better Health in Aging Societies - Proceedings of MIE 2006, The XXst International Congress of the European Federation for Medical Informatics, Maastricht, The Netherlands, August 27-30, 2006*, volume 124 of *Studies in Health Technology and Informatics*, pages 157–161. IOS press, 2006.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, D Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[BCR08] Alexander Borgida, Diego Calvanese, and Mariano Rodriguez. Explanation in dl-lite. In *Description Logics*, 2008.

[BEL01] Matthias Baaz, Uwe Egly, and Alexander Leitsch. Normal form transformations. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, pages 273–333. Elsevier and MIT Press, 2001.

[BFH00] Alex Borgida, Enrico Franconi, and Ian Horrocks. Explaining $\mathcal{ALC}$ subsumption. In Werner Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 209–213. IOS Press, 2000.

[BG09] Dan Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium, February 2009.

[BGB$^+$99] Patricia G. Baker, Carole A. Goble, Sean Bechhofer, Norman W.

Paton, Robert Stevens, and Andy Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999.

[BH95] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14(1):149–180, 1995.

[BHGS01] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: A Reason-able ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, number 2174 in Lecture Notes in Artificial Intelligence, pages 396–408. Springer, 2001.

[Bie08] M. Bienvenu. Prime implicate normal form for $\mathcal{ALC}$ concepts. In *AAAI-08*, pages 412–417, 2008.

[Bie09] Meghyn Bienvenu. Prime implicates and prime implicants: From propositional to modal logic. *Journal of Artificial Intelligence Research*, 36:71–128, 2009.

[BLS06a] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.

[BLS06b] F. Baader, C. Lutz, and B. Suntisrivaraporn. Efficient reasoning in $\mathcal{EL}^+$. In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, CEUR-WS, 2006. To appear.

[BMC03] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG description logic interface. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *Proceedings of the 2003 International Workshop on Description Logics (DL 2003), Rome, Italy, September 5–7, 2003*, volume 81 of *CEUR Workshop Proceedings*. CEUR, September 2003.

[BP10] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic Computation*, 20(1):5–34, 2010.

[BPS07] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}$. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany*, volume 4667 of *Lecture Notes in Computer Science*, pages 52–67. Springer, September 2007.

[Bra04] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In R. López de Mantáras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, pages 298–302. IOS Press, 2004.

[BS01] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[BS05] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of contraints using hitting set dualization. In *Practical Aspects of Declarative Languages (PADL 05)*, 2005.

[BS08] Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the 3rd Knowledge Representation in Medicine Conference (KR-MED'08): Representing and Sharing Knowledge Using SNOMED*, 2008.

[BVL03] Sean Bechhofer, Raphael Volz, and Philip Lord. Cooking the semantic web with the OWL API. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003. The Second International Semantic Web Conference, Sanibel Island, Florida, USA*, volume 2870/2003 of *Lecture Notes in Computer Science*, pages 659–675, Sanibel Island, Florida, USA, October 2003. Springer.

[BvW06] Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors. *The Handbook of Modal Logic*, volume 3 of *Studies in Logic*. Elsevier, 2006.

[CGL97] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment in description logics with n-ary relations. In Ronald J. Brachman, Francesco M. Donini, Enrico Franconi, Ian Horrocks, Alon Y. Levy, and Marie-Christine Rousset, editors, *Proceedings of the 1997 International Workshop on Description Logics, Université Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique LRI*, volume 410 of *URA-CNRS*, 1997.

[Chi97] John W. Chinneck. Finding a useful subset of constraints for analysis in an infeasible linear program. *Informs Journal on Computing*, 9:164–174, 1997.

[CHKS07] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Just the right amount: Extracting modules from ontologies. In *WWW 2007, Proceedings of the 16th International World Wide Web Conference, Banff, Canada, May 8-12, 2007*, pages 717–727, 2007.

[CHM$^+$08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 2008.

[CHWK07] Bernardo Cuenca Grau, Christian Halasheck-Wiener, and Yevgeny Kazakov. History matters: Incremental ontology reasoning using modules. In Karl Aberer, Key-Sun Choi, Natalya F. Noy, Guus Schreiber, and Riichiro Mizoguchi, editors, *The Semantic Web - 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 183–196. Springer, 2007.

[CLLR05] Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Dl-lite: Tractable description logics for ontologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 602–607, 2005.

[Coc50] William Gemmell Cochran. The comparison of percentages in matched samples. *Biometrika*, 37:256–266, 1950.

[DFK$^+$07] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma. Scalable semantic retrieval through summarization and refinement. In Robert C. Holte and Adele Howe, editors, *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 299–304. AAAI Press, 2007.

[DH88] Randall Davis and Walter C. Hamscher. Model-based reasoning: troubleshooting. *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, pages 297–346, 1988.

[dKAHC$^+$99] N. F. de Keizer, Ameen. Abu-Hanna, Ronald Cornet, Johanna H. M. Zwetsloot-Schonk, and Chris P. Stoutenbeek. Analysis and design of an ontology for intensive care diagnoses. *Methods of Information in Medicine*, 38:102–112, 1999.

[dKBRJC08] N. F. de Keizer, F Bakhshi-Raiez, E De Jonge, and Ronald Cornet. Post-coordination in practice: evaluating compositional terminological-system-based registration of icu reasons for admission. *International Journal of Medical Informatics*, 77:828–835, 2008.

[DLR08] P. P. Kanjamala Mark A. Musen Daniel L. Rubin, Dilvan de Abreu Moreira. BioPortal: A web portal to biomedical ontologies. In Peter Fox Deborah L. McGuinness and Boyan Brodaric, editors, *AAAI Spring Symposium Serires, Symbiotic Relationships between Semantic Web and Knowledge Engineering, Stanford University*, volume Technical Report SS-08-05, page 136. AAAI Press, March 2008.

[dSSC$^+$08] Paulo Pinheiro da Silva, Geoff Sutcliffe, Cynthia Chang, Li Ding, Nick del Rio, and Deborah L. McGuinness. Presenting tstp proofs with inference web tools. In *PAAR/ESHOL*, 2008.

[ELM$^+$05] Karen Eilbeck, Suzanna Lewis, Christopher J. Mungall, Mark Yandell, Lincoln Stein, Richard Durbin, and Michael Ashburner. The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biology*, 6(5), April 2005.

[End00] Ulrich Endriss. Reasoning in description logics with wellington 1.0 – system description. In *Proceedings of the 7th Workshop on Automated Reasoning. Bridging the Gap between Theory and Practice*, volume 32 of *CEUR Workshop Proceedings*. CEUR-WS.org, July 2000.

[FS05] Gerhard Friedrich and Kostyantyn Shchekotykhin. A general diagnosis method for ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web – ISWC 2005 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 232–246. Springer, October 2005.

[G$\ddot{9}$2] Peter Gärdenfors. *Belief Revision: An Introduction*, pages 1–20. Cambridge University Press, 1992.

[Gen84] Michael R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1):411–436, December 1984.

[GFH$^+$03] Jennifer Golbeck, Gilberto Fragoso, Frank Hartel, Jim Hendler, Jim Oberthaler, and Bijan Parsia. National cancer institute's thesaurus and ontology. *Journal of Web Semantics*, 1(1):75–80, December 2003.

[GH07] Christine Golbreich and Ian Horrocks. The OBO to OWL mapping, GO to OWL 1.1! In *Proc. of the Third OWL Experiences and Directions Workshop*, number 258 in CEUR (`http://ceur-ws.org/`), 2007.

[GHH$^+$07] Christine Golbreic, Matthew Horridge, Ian Horrocks, Boris Motik, and Rob Shearer. OBO and OWL: Leveraging semantic web technologies for the life sciences. In Karl Aberer, Key-Sun Choi, Natalya F. Noy, Guus Schreiber, and Riichiro Mizoguchi, editors, *The*

Semantic Web - 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007, volume 4825 of *Lecture Notes in Computer Science*, pages 169–182. Springer, 2007.

[GHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research (JAIR)*, 31:273–318, 2008.

[GiL99] Ontology for immunogenetics: the IMGT-ONTOLOGY. *Bioinformatics*, 15(12):1047–1054, June 1999.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Books in the Mathematical Sciences. W. H. Freeman, January 1979.

[GLW06] Silvio Ghilardi, Carsten Lutz, and Frank Wolter. Did i damage my ontology? a case for conservative extensions in description logic. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *The 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District, United Kingdom.* AAAI Press, June 2006.

[Gro09] Mike Grove. OWLSight. `http://pellet.owldl.com/ontology-browser`, October 2009.

[GRV10] Birte Glimm, Sebastian Rudolph, and Johanna Völker. Integrated metamodeling and diagnosis in OWL 2. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang 0007, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes In Computer Science*, pages 257–272. Springer, 2010.

[GSW89] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41:79–88, 1989.

[GW11] Stephan Grimm and Jens Wissmann. Elimination of redundancy in ontologies. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dmitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semantic Web: Research and Applications. 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes In Computer Science*, pages 260–274. Springer, June 2011.

[Hal77] Maurice H. Halstead. *Elements of Software Science*. Elsevier, New York, 1977.

[HB09] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for working with OWL 2 ontologies. In Rinke Hoeksta and Peter F. Patel-Schneider, editors, *OWL: Experiences and Directions (OWLED 2009), 6th OWL Experienced and Directions Workshop, Chantilly, Virginia, October 2009.*, CEUR Workshop Proceedings. CEUR, October 2009.

[HB11] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, February 2011.

[HBPS08] Matthew Horridge, Johannes Bauer, Bijan Parsia, and Ulrike Sattler. Understanding entailments in OWL. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2008.

[HBPS11a] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyashev, editors, *Proceedings of the 24th International Workshop on Description Logics (DL2011), Barcelona, Spain July 13–16, 2011*, CEUR Workshop Proceedings. CEUR-WS.org, July 2011.

[HBPS11b] Matthew Horridge, Samantha Bail, Bijan Parsia, and Ulrike Sattler. The cognitive complexity of OWL justifications. In Christopher A. Welty, Lora Aroyo, and Natalya F. Noy, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, ISWC 2011, Bonn, Germany, October 23-27, 2011*, Lecture Notes In Computer Science. Springer, 2011.

[HCK92] Walter Hamscher, Luca Console, and Johan de Kleer, editors. *Readings in Model Based Diagnosis*. Morgan Kaufmann Publishers Inc., June 1992.

[HDG$^+$06] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H Wang. The manchester owl syntax. In *OWL: Experiences and Directions (OWLED)*, 2006.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *The 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District, United Kingdom*, pages 57–67. AAAI Press, June 2006.

[HM01] Volker Haarslev and Ralf Möller. RACER system description. In *International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes In Computer Science*, pages 701–705, 2001.

[Hor97] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[Hor02] Ian Horrocks. DAML+OIL: a reason-able web ontology language. In *Proc. of EDBT 2002*, number 2287 in Lecture Notes in Computer Science, pages 2–13. Springer, March 2002.

[Hor05] Ian Horrocks. Applications of description logics: State of the art and research challenges. In Frithjof Dau, Marie-Laure Mugnier, and Gerd Stumme, editors, *Conceptual Structures: Common Semantics for Sharing Knowledge, 13th International Conference on Conceptual Structures, ICCS 2005, Kassel, Germany, July 17-22,* *2005, Proceedings*, volume 3596 of *Lecture Notes In Computer Science*, pages 78–90. Springer, 2005.

[HP10] Matthew Horridge and Bijan Parsia. From justifications towards proofs for ontology engineering. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010.

[HPS08a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explanation of OWL entailments in Protégé-4. In Christian Bizer and Anupam Joshi, editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2008.

[HPS08b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web – ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008.ISWC 2008*, volume 5318 of *Lecture Notes In Computer Science*, pages 323–338. Springer, October 2008.

[HPS08c] Matthew Horridge and Peter F. Patel-Schneider. Manchester OWL Syntax for OWL 1.1. In *OWL: Experiences and Directions (OWLED)*, 2008.

[HPS09a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Computing explanations for entailments in Description Logic based ontologies. In *16th Automated Reasoning Workshop (ARW 2009), Liverpool, UK.*, 2009.

[HPS09b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explaining inconsistencies in OWL ontologies. In Luis Godo and Andrea

Pugliese, editors, *3rd International Conference on Scalable Uncertainty Management SUM 2009, September 28–30, 2009 Washington DC Area, USA*, volume 5785 of *Lecture Notes In Computer Science*, pages 124–137. Springer, 2009.

[HPS09c] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. From justifications to proofs for entailments in OWL. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[HPS09d] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in OWL. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics (DL 2009)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, July 2009.

[HPS10a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification masking in OWL. In Grant Weddell, Volker Haarslev, and David Toman, editors, *Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Canada. May 4th–May 7th, 2010*, 2010.

[HPS10b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification oriented proofs in OWL. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes In Computer Science*, pages 354–369. Springer, November 2010.

[HPS11] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The state of biomedical ontologies. In *BioOntologies 2011 Co-Located with ISMB 2011, 15th–16th July, Vienna Austria*, 2011.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen.

From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[HS07] Ian Horrocks and Ulrike Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, 2007.

[HST00] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3), 2000.

[HTR06] Matthew Horridge, Dmitry Tsarkov, and Timothy Redmond. Supporting early adoption of OWL 1.1 with Protégé-OWL and FaCT++. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWL: Experiences and Directions (OWLED)*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, November 2006.

[Hua94] Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, editor, *Automated Deduction - CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 - July 1, 1994*, volume 814 of *Lecture Notes In Computer Science*, pages 738–752. Springer, July 1994.

[Hv06] Zhisheng Huang and Frank van Harmelen. Reasoning with inconsistent ontologies: Evaluation. Sekt ed-ist-2003-506826 deliverable, Vrije Universiteit Amsterdam, January 2006.

[HWKP06] Christian Halaschek-Wiener, Yarden Katz, and Bijan Parsia. Belief base revision for expressive description logics. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the OWLED 06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11, 2006.*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[JE10] Dietmar Jannach and Ulrich Engler. Toward model-based debugging of spreadsheet programs. In Alberta Caplinskas and Takako Nakantani, editors, *9th Joint Conference on Knowledge-Based Software Engineering (JCKBSE 10) August 25-27, 2010, Kaunas, Lithuania*, pages 252–264, August 2010.

[JHQ$^+$09] Qiu Ji, Peter Haase, Guilin Qu, Pascal Hitzler, and Steffen Stadtmoeller. RaDON – repair and diagnosis in ontology networks. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Simperl, editors, *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009 Heraklion, Crete, Greece, May 31–June 4, 2009*, volume 5554/2009, pages 863–867. Springer Berlin /Heidelberg, May 2009.

[JLB91] Philip N. Johnson-Laird and Ruth M. J. Byrne. *Deduction.* Psychology Press, 1991.

[JQH09] Qiu Ji, Guilin Qi, and Peter Haase. A relevance-directed algorithm for finding justifications of dl entailments. In Asunción Gómez-Pérez, Yong Yu, and Ying Ding, editors, *The Semantic Web, Fourth Asian Conference, ASWC 2009, Shanghai, China, December 6-9, 2009. Proceedings*, volume 5926 of *Lecture Notes In Computer Science*, pages 306–320. Springer, December 2009.

[Jun01] Ulrich Junker. QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms. In Christian Bessiere, Francois Laburthe, Pedro Meseguer, Jean-Charles Regin, Francesca Rossi, Babara Smith, and Toby Walsh, editors, *Workshop on Modelling and Solving Problems with Constraints, CONS-1, August 2001. nternational Joint Conference on Artificial Intelligence (IJCAI-2001), Seattle WA*, August 2001.

[Jun04] Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In Anthony G. Cohn, editor, *Proceedings of the 19th national conference on Artifical intelligence (AAAI 04) San Jose, California,*, pages 167–172. AAAI Press, 2004.

[Kal06] Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies.* PhD thesis, The Graduate School of the University of Maryland, 2006.

[Kaz09] Yevgeny Kazakov. Consequence-driven reasoning for horn $\mathcal{SHIQ}$ ontologies. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *22nd International Workshop on Description Logics, Oxford, UK*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, July 2009.

[KFNM04] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In Sheila McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *ISWC 04 The International Semantic Web Conference 2004, Hiroshima, Japan*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[KK07] Petr Kremen and Zdenek Kouba. Incremental approach to error explanations in ontologies. In *I-KNOW 07. Graz: Graz University of Technology, 2007*, pages 332–339, 2007.

[Knu07] Holger Knublauch. Composing the semantic web: Explaining inferences. `http://composing-the-semantic-web.blogspot.com/2007/08/explanining-inferences.html`, August 2007.

[KPG06] Aditya Kalyanpur, Bijan Parsia, and Bernardo Cuenca Grau. Beyond asserted axioms: Fine-grain justifications for OWL-DL entailments. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *DL 2006, Lake District, U.K.*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[KPH05] Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. In *International Journal on Semantic Web and Information Systems*, volume 1, Jan - Mar 2005.

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In Karl Aberer, Key-Sun Choi, Natalya F. Noy, Guus Schreiber, and Riichiro Mizoguchi, editors, *The Semantic Web - 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2007.

[KPS05] Aditya Kalyanpur, Bijan Parsia, and Evren Sirin. Black box techniques for debugging unsatisfiable concepts. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL2005), Edinburgh, Scotland, UK, July 26-28, 2005*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, July 2005.

[KPSG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in OWL ontologies. In *European Semantic Web Conference (ESWC), Budva, Montenegro 2006*, 2006.

[KPSH05] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4), 2005.

[Kwo05] Francis King Hei Kwong. Practical approach to explaining $\mathcal{ALC}$ subsumption. Technical report, The University of Manchester, 2005.

[Lam07] Sik Chun Joey Lam. *Methods for Resolving Inconsistencies In Ontologies*. PhD thesis, Department of Computer Science, Aberdeen, 2007.

[LBF$^+$06] Carsten Lutz, Franz Baader, Enrico Franconi, Domenico Lembo, Ralf Möller, Riccardo Rosati, Ulrike Sattler, Boontawee Suntisrivaraporn, and Sergio Tessaris. Reasoning support for ontology design. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the OWLED 06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, Novemver 10–11, 2006*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, November 2006.

[Lew82] Clayton H. Lewis. Using the thinking-aloud method in cognitive interface design. Research report RC-9265, IBM, 1982.

[Lin89] Christoph Lingenfelder. Structuring computer generated proofs. In *proceedings of the Eleventh International Joint Conference on Artificial Intelligence, August 20-25, 1989, Detroit, Michigan, USA, Volume 1*, pages 378–383. Morgan Kaufmann, 1989.

[LM05] Kevin Lee and Thomas Meyer. A classification of ontology modification. In Geoffrey Webb and Xinghuo Yu, editors, *AI 2004: Advances in Artificial Intelligence*, volume 3339 of *Lecture Notes in Computer Science*, pages 181–214. Springer Berlin / Heidelberg, 2005.

[LMPB06] Kevin Lee, Thomas Meyer, Jeff Z. Pan, and Richard Booth. Computing maximally satisfiable terminologies for the description logic $\mathcal{ALC}$ with cyclic definitions. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL2006), Windermere, Lake District, UK May 30 – June 1, 2006*, volume 189 of *CEUR Workshop Proceedings*. CEUR, June 2006.

[LN04] Thorsten Liebig and Olaf Noppens. OntoTrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. In Sheila McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web - ISWC 2004. Third International Semantic Web Conference 2004, Hiroshima, Japan*, volume 3298 of *Lecture Notes in Computer Science*, pages 244–258. Springer, November 2004.

[LPSV06] Sik Chun Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. In Jiming Liu, Benjamin W. Wah, and Toyoaki Nishida, editors, *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI'06), 18 - 22 December 2006, Hong Kong, China*, pages 428–434, Los Alamitos, CA, USA, December 2006. IEEE Computer Society.

[LSPV08] Joey Sik Chun Lam, Derek H. Sleeman, Jeff Z. Pan, and Wamberto Weber Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. *Journal of Data Semantics*, 10:62–95, 2008.

[MBB95] Deborah L. McGuinness, Alexander T. Borgida, and Er T. Borgida. Explaining subsumption in description logics. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal, Québec, Canada*, volume 1, pages 816–821. Morgan Kaufmann Publishers Inc., August 1995.

[MBB$^+$01] Chris J. Mungall, Michael Bada, Tanya Z. Berardini, Jennifer Deegan, Amelia Ireland, Midori A. Harris, David P. Hill, and Jane Lomax. Cross-product extensions of the gene ontology. *Journal of Biomedical Informatics*, 44(1):80–86, February 2001.

[MBE11] Christopher J. Mungall, Colin Batchelor, and Karen Eilbeck. Evolution of the sequence ontology terms and relationships. *Journal of Biomedical Informatics*, 44:87–93, February 2011.

[McC76] Thomas J. McCabe. A complexity measure. In *IEEE Transactions On Software Engineering*, volume SE-2 of *2*, pages 308 – 320, December 1976.

[McG96] Deborah L. McGuinness. *Explaining Reasoning in Description Logics.* PhD thesis, Rutgers University Department of Computer Science, 1996.

[McN47] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. 10.1007/BF02295996.

[MHA$^+$10] James Malone, Ele Holloway, Tomasz Adamusiak, Misha Kapushesky, Jie Zheng, Nikolay Kolesnikov, Anna Zhukova, Alvis Brazma, and Helen Parkinson. Modelling sample variables with and experimental factor ontology. *Bioinformatics*, 26(8):1112–1118, March 2010.

[MHL07] Yue Ma, Pascal Hitzler, and Zuoquan Lin. Algorithms for paraconsistent reasoning with OWL. In *ESWC*, volume 4159, pages 399–413, June 2007.

[Min75] Marvin Minsky. A framework for representing knowledge. In *The Psychology of Computer Vision*, 1975.

[MLBP06] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *21st National Conference on Artificial Intelligence, AAAI*, 2006.

[MMV10] Thomas Meyer, Kodylan Moodley, and Ivan Varzinczak. First steps in the computation of root justifications. In Alan Bundy, Jos Lehmann, Guilin Qi, and Ivan Varzinczak, editors, *2nd International Workshop on Automated Reasoning about Context and Ontology Evolution (ARCOE), 16–17 August 2010, Lisbon, Portugal*, 2010.

[Moo10] Kodylan Moodley. Debuging and repair of description logic ontologies. Master's thesis, School of Computer Science, University of KwaZulu-Natal, Durban, December 2010.

[Mot06] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.

[MPS98] Deborah L. McGuinness and Peter F. Patel-Schneider. Usability issues in knowledge representation systems. In Jack Mostow and Charles Rich, editors, *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 608–614, Menlo Park, CA, USA, July 1998. American Association for Artificial Intelligence.

[MPSP09] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language structural specification and functional style syntax. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.

[MRBP08] James Malone, Tim F. Rayner, Xiangqun Zheng Bradley, and Helen Parkinson. Developing an application focused experimental factor ontology: embracing the OBO community. In Philip Lord, Susanna-Assunta Sansone, Nigam Shah, and Matt Cockerill, editors, *16th Annual International Conference on Intelligent Systems for Molecular Biology, 2008 SIG Meeting on Bio-Ontologies, Toronto, Canada*, July 2008.

[MS06] Boris Motik and Ulrike Sattler. Practical DL reasoning over large ABoxes with KAON2. In *Principles of Knowledge Representation and Reasoning*, CEUR Workshop Proceedings. CEUR-WS.org, 2006.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 67–83. Springer, 2007.

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[Mun11] Chris Mungall. OBO Flat File Format 1.4 syntax and semantics. ftp://ftp.geneontology.org/pub/go/www/obo-syntax.html, February 2011.

[Mur82] Neil V. Murray. Completely non-clausal theorem proving. *Artificial Intelligence*, 18(1):67 – 85, 1982.

[nal09] Rafael Pe naloza. *Axiom-Pinpointing in Description Logics and Beyond.* PhD thesis, Dresden University of Technology, 2009.

[NBH$^+$06] Stephen E. Newstead, Peter Brandon, Simon J. Handley, Ian Dennis, and Jonathan St. B. Evans. *Predicting the Difficult of Complex Logical Reasoning Problems*, volume 12. Psychology Press, 2006.

[NBM09] Riku Nortje, Katarina Britz, and Thomas Meyer. Finding $\mathcal{EL}^+$ justifications using the Earley parsing algorithm. In Thomas Meyer and Kerry Taylor, editors, *Advances in Ontologies. Procededings of the Fifth Australasian Ontology Workshop, Melbourne, Australia, December 2009*, December 2009.

[NDG$^+$09] Natalya F. Noy, Michael V. Dorf, Nicholas Griffith, Csongor Nyulas, and Mark A. Musen. Harnessing the power of the community in a library of biomedical ontologies. In Tim Clark, Joanne S. Luciano, M. Scott Marshall, Eric Prud'hommeaux, and Susie Stephens, editors, *Proceedings of the Workshop on Semantic Web Applications in Scientific Discourse (SWASD 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington DC, USA, October 26, 2009.*, volume 523 of *CEUR Workshop Proceedings*. CEUR-WS.org, November 2009.

[Neb90] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, 1990.

[NM03] Natalya F. Noy and Mark A. Musen. The PROMPT suite: interactive tools for ontology mergine and mapping. *International Journal of Human Computer Studies*, 59(6):983–1024, December 2003.

[Nor01] Riku Nortje. Module extraction for inexpressive description logics. Master's thesis, University of South Africa, February 2001.

[NSW^{+}09] Natalya F. Noy, Nigam H. Shah, Patrisha L. Whetzel, Benjamin Dai, Michael V. 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, May 2009.

[NW01] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 335–367. Elsevier and MIT Press, 2001.

[PG86] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.

[Pot74] George R. Potts. Storing and retrieving information about ordered relationships. *Journal of Experimental Psychology*, 103(3):431 – 439, 1974.

[Pri02] Graham Priest. *Handbook of Philosophical Logic*, volume 6, chapter Paraconsistent Logic, pages 287–393. Kluwer Academic Publishers, 2nd edition, 2002.

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 20*, pages 633–640. Association for Computing Machinery, Inc. (ACM), May 2005.

[Qui52] Willard Van Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, October 1952.

[Qui55] Willard Van Quine. A way to simplify truth functions. *The American Mathematical Monthly*, 62(9):627–631, November 1955.

[Qui59] Willard Van Quine. On cores and prime implicants of truth functions. *The American Mathematical Monthly*, 66(9):755–760, November 1959.

[RBG$^+$97] Alan L. Rector, Sean Bechhofer, Carole Goble, Ian Horrocks, Anthony W. Nowlan, and Daniel Soloman. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.

[RCVB09] Catherine Roussey., Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of OWL ontology AntiPatterns. In Natalya F. Noy and Yolanda Gil, editors, *K-CAP 2009 – Proceedings of the 5th International Conference on Knowledge Capture, September 1–4, 2009, Redondo Beach, California, USA*, pages 205–206, 2009.

[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. In *14th International Conference on Knowledge Engineering and Knowledge Management EKAW 2004, 5-8th October 2004 - Whittlebury Hall, Northamptonshire, UK*, pages 63–81, October 2004.

[Rec03] Alan L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In John

Gennari and Bruce Porter, editors, *K-CAP 2003 – Proceedings of the 2nd international conference on Knowledge Capture, October 23–25, 2003, Sanibel Island, Florida, USA*, K-CAP '03, pages 121–128, New York, NY, USA, 2003. ACM.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1):81–132, 1980.

[Rei87] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Rip94] L. J. Rips. *The Psychology of Proof.* MIT Press, Cambridge, MA, 1994.

[RJ03] Cornelius Rosse and José L. V. Mejino Jr. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics*, 36(6):478–500, 2003.

[RN10] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

[RNG93] Alan L. Rector, Anthony W. Nowlan, and Andrzej Glowinski. Goals for concept representation in the GALEN project. In *In Proceedings of the 17th Annual Symposium on Computer Applications in Medical Care, October 30 – November 3, 1993, Washington DC*, American Medical Informatics Association, pages 414–418, 1993.

[SAR$^+$07] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J. Mungall, Neocles Leontis, Phillipe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H. Scheuermann, Nigam Shah, Patrisha L. Whetzel, and Suzanna Lewis. The OBO Foundary: Coordinated evolution of ontology to support biomedical data integration. *Nature Biotechnology*, 25:1251–1255, November 2007.

[Sat96] Ulrike Sattler. A concept language extended with different kinds of transitive roles. In Günther Görz and Steffen Hölldobler, editors, *KI-96: Advances in Artificial Intelligence, 20th Annual German Conference on Artificial Intelligence, Dresden, Germany*, volume 1137 of *Lecture Notes in Computer Science*, pages 333–345. Springer, September 1996.

[SC97] Kent A. Spackman and Keith E. Campbell. SNOMED RT: A reference terminology for health care. In Daniel R. Masys, editor, *Proceedings of AMIA Annual Fall Symposium*, pages 640–644, Bethesda, Maryland, USA, October 1997. Hanley and Belfus Inc.

[SC03] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 355–362. Morgan Kaufmann, 2003.

[Sch04] Stefan Schlobach. Explaining subsumption by optimal interpolation. In José Júlio Alferes and João Alexandre Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30-2004*, volume 3229 of *Lecture Notes in Computer Science*, pages 413–425. Springer, September 2004.

[Sch05a] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In Asunción Gómez-Pérez and Jerome Euzenat, editors, *The Semantic Web: Research and Applications, 2nd European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29–June 1, 2005, Proceedings*, volume 3532 of *Lecture Notes In Computer Science*, pages 226–240. Springer-Verlag Berlin Heidelberg, May 2005.

[Sch05b] Stefan Schlobach. Diagnosing terminologies. In *AAAI'05: Proceedings of the 20th national conference on Artificial intelligence*, pages 670–675. AAAI Press, 2005.

[SCH10] Giorgos Stoilos, Bernardo Cuenca Grau, and Ian Horrocks. How incomplete is your semantic web reasoner? In Nestor Rychtyckyj, Daniel Shapiro, Maria Fox, and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, July, 2010 Atlanta, Georgia USA*. AAAI Press, July 2010.

[SdF+10] Thomas Scharrenbach, Claudia d'Amato, Nicola Fanizzi, Rolf Grütter, Bettina Waldvogel, and Abraham Bernstein. Default Logics for Plausible Reasoning with Controversial Axioms. In Fernando Bobillo, editor, *Proceedings of the 6th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW-2010), 7th November, 2010, Shanghai, China*, CEUR Workshop Proceedings. CEUR Workshop Proceedings, November 2010.

[SFJ08] Kostyantyn Shchekotykhin, Gerhard Friedrich, and Dietmar Jannach. On computing minimal conflicts for ontology debugging. In Bernhard Peischl, Neil Snooke, Gerald Steinbauer, and Cees Witteveen, editors, *ECAI 2008 Workshop on Model-Based Systems, July 21-22, Patras, Greece. Affiliated with the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 7–11, 2008.

[SH07] Stefan Schlobach and Zhisheng Huang. Inconsistent ontology diagnosis and repair. SEKT ED-IST-2003-506826 Deliverable SEKT/2006/D3.6.3/v1.0.0, Vrije Universiteit Amsterdam, February 2007.

[SHCvH07] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39:317 – 349, 2007.

[SPG+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), 2007.

[SQJH08] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A modularization-based approach to finding all justifications for owl dl entailments. In John Domingue and Chutiporn Anutariya, editors, *Proceedings of the 3th Asian Semantic Web Conference (ASWC'08)*, volume 5367 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2008.

[SR06] Julian Seidenberg and Alan L. Rector. Web ontology segmentation: analysis, classification and use. In Leslie Carr, David De Roure, Arun Iyengar, Carole Goble, and Mike Dahlin, editors, *Proceedings of the 15th international c*, pages 13–22, New York, NY, USA, 2006. Association for Computing Machinery, Inc. (ACM).

[SSS91] Manfred Schmidt-Schauss and Gert Smolka. Attributive concept descreiptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[SSZ09] Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which kind of module should i extract? In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[Str92] Gerhard Strube. The role of cognitive science in knowledge engineering. In *Proceedings of the First Joint Workshop on Contemporary Knowledge Engineering and Cognition*, pages 161–174, London, UK, 1992. Springer-Verlag.

[Stu08] Heiner Stuckenschmidt. Debugging OWL ontologies - a reality check. In Raul Garcia-Castro, Asunción Gómez-Pérez, Charles J. Petrie, Emanuele Della Valle, Ulrich Küster, Michal Zaremba, and M. Omair Shafiq, editors, *EON-SWSC 2008 Proceedings of the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge, Tenerife, Spain, June, 2008*, volume 359 of *CEUR Workshop Proceedings*. CEUR-WS.org, June 2008.

[Sun08] Boontawee Suntisrivaraporn. Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}^+$ ontologies. In Sean Bechhofer, Manfred Hauswirth, Joerg Hoffmann, and Manolis Koubarakis, editors, *Proceedings of the 5th European Semantic Web Conference (ESWC'08)*, volume 5021 of *Lecture Notes in Computer Science*, pages 230–244. Springer-Verlag, 2008.

[Sun09] Boontawee Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies.* PhD thesis, Technical University of Dresden, 2009.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[Top09] TopQuadrant. Topquadrant: Products: TopBraid Composer. `http://www.topquadrant.com/products/TB_Composer.html`, October 2009.

[Tse68] G. S. Tseitin. On the complexity of derivations in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, 1968.

[WAH$^+$07] Katy Wolstencroft, Pinar Alper, Duncan Hull, Chris Wroe, Phillip W. Lord, Robert D. Stevens, and Carole A. Goble. The $^{\text{my}}$Grid ontology: bioinformatics service discovery. *International Journal of Bioinformatics Resesearch and Applications*, 3(3):303–325, 2007.

[Wey88] Elaine J. Weyuker. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, 1988.