MODULE-BASED CLASSIFICATION OF OWL ONTOLOGIES

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

2016

By
Nicolas Matentzoglu
School of Computer Science

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I want to thank my amazing supervisors Bijan Parsia and Uli Sattler. Your doors were always open, your patience with my slow grasp of logic seemingly endless and your lessons went far beyond what made it into these pages. In the end, you both made these years the most intense, and rewarding, learning experience of my life.

I want to thank my examiners, Bernardo Cuenca Grau and Graham Riley for the interesting viva and the valuable suggestions to improve this thesis.

Next, I want to thank the folks in our group. Leo, you have saved me thousands of times when I reached my limit with $\exists$ and $\forall$, but most importantly, you were a good friend. Sam and Rafa, you have taught me a great deal about experiments and OWL, and helped me do my first steps. I also thank Valentino, Michael, Slava, Chiara, Eleni, Tahani and Colin for many discussions and advice. Ignazio and Dima, you helped me a great deal sorting out my OWL API and reasoner problems, Dima especially with proof-reading. Thanks a lot.

I also want to thank my CDT gang, Aitor, Tom, Dave, Colin, Kostas, Ed, Ilia, Michele, Fardeen, Rob and Matt. We had great times together, and our cohort was definitely the one and only real one.

A big thanks also to my family, Maria, Silvia and Simeon, and my friends, Theo, Stephan and many more, for always supporting me.

Last, but not least, I want to say a big thank you to my beloved kween Maria. To your beauty and all the amazing moments we shared, and will continue to share.

I acknowledge and thank the UK Engineering and Physical Science Research Council (EPSRC) for its support in the form of a doctoral training grant.

Abstract

Module-based classification of OWL ontologies
Nicolas Matentzoglu
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2016

Classification is a core reasoning service provided by most OWL reasoners. Classification in general is hard—up to 2NExptime for $\mathcal{SROIQ}(\mathcal{D})$, the Description Logic which is underpinning the Web Ontology Language (OWL). While it has been shown that classification is practical for a wide range of inputs, there are still ontologies for which classification takes an unreasonable amount of time for purposes such as ontology engineering (frequent classifications after updates).

A natural optimisation strategy is divide and conquer, that is, to decompose the ontology into subsets which are hopefully easier to classify and whose classifications can be combined into a complete classification of the whole ontology. Unfortunately, an arbitrary subset may not be self-contained, i.e. it might be missing information that is needed to determine entailments over its signature. Moreover, such a subset can be potentially harder to classify than the whole ontology. In order to mitigate those problems, classification preserving decompositions (CPDs) must be designed with care that they support complete classification which is, in practice, more efficient than monolithic classification. Locality-based modules are subsets of an ontology that provide certain guarantees with respect to the entities (concepts, roles) in its signature—in particular, modules are self-contained.

In this thesis we explore the use of syntactic locality-based modules for underpinning classification-preserving decompositions. In particular, we empirically explore their potential to avoid subsumption tests and reduce subsumption test hardness and weigh those benefits against detrimental effects such as overhead (for example the time it takes to compute the decomposition) and redundancy (a consequence of potentially overlapping chunks in the decomposition). The main contributions of this thesis are an in-depth empirical characterisation of these

effects, an extensible framework for observing CPDs in action up until a granularity of individual subsumption tests, a large, public corpus of observations and its analysis and insights on experimental methodologies around OWL reasoning.

Declaration

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

Copyright

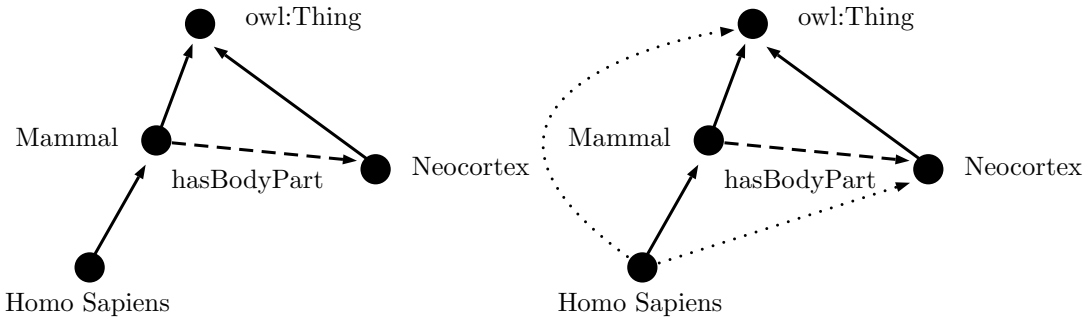
Chapter 1

Introduction

1.1 OWL Reasoning

An ontology is a computationally accessible conceptualisation of a domain, formally specifying the entities in that domain and their interrelations. Ontologies and ontological systems are considered core areas of research in artificial intelligence and knowledge representation [PCMB15], especially in domains with a large amount of declarative knowledge such as medicine and biology [NSW+09, Con13, RAK06]. Some ontologies now constitute key parts in the tool chain of domain experts: The Gene Ontology (GO) has become a standard means to unify the representation of knowledge on genes across species [Con13]; SNOMED captures complex clinical knowledge for use in medical information systems and clinical research [RAK06]; the upcoming version of the International Classification of Diseases (ICD-11) comes with a formally defined core ontology [TNNM13]. BioPortal, a repository for bio-health ontologies, hosts more than 400 biomedical ontologies [NSW+09] and Ontohub, and open ontology repository, claims to host more than 3000 ontologies ranging from toy examples to production quality ontologies [MKC14].

Despite these indicators for success and being a research theme for almost three decades [BCM+07], harnessing the full potential of ontologies is still a *work in progress*. The deployment of ontologies for use in applications (and thus extensive API and tool development) grew significantly after the standardisation of the Web Ontology Language (OWL) around 10 years ago. As an emerging technology, the uptake of OWL ontologies to solve industrial scale problems progresses rather hesitantly, compared to the strong academic interest in the subject. Especially the expectations of interested industrial groups are often frustrated. Some of the frequently mentioned bottlenecks for uptake are inadequate tool support, limitations in expressive power, lack of methodologies (both for ontology development and the evaluation of ontological systems) and a lack of educational



(a) Asserted structure. (b) Tacit (implicit) knowledge.

Figure 1.1: An example of asserted and tacit (or implicit) knowledge in the ontology. Because the reasoner understands the semantics of OWL, it can infer from the fact that a human is a mammal and mammals have a neocortex, that humans also have a neocortex.

resources to compensate the difficulty of ontological modelling.

In this thesis, we are concerned with a key tool for ontology development and ontology-based applications: *OWL reasoners*. Deriving implicit knowledge from the asserted knowledge in ontologies is referred to as *OWL reasoning*, or reasoning in short. Reasoning enables users to query all knowledge in the ontology and verify the consistency and coherence of an ontology; for a simple example inference see Figure 1.1. OWL reasoners are the software tools used for performing reasoning tasks. The potential for providing reasoning services that are sound, complete and terminating[1] served as the primary motivation to design ontology languages that are underpinned by Description Logic Semantics [GHM+08]. The family of Description Logic languages are well understood fragments of First Order Logic. OWL in particular is underpinned by the Description Logic $\mathcal{SROIQ}$ [HKS06].

In this thesis we will focus in particular on the key reasoning service of classification, that is, computing the subsumption relation between all named concepts in the ontology. This service is particularly important for two groups of people: Knowledge engineers, who are building and maintaining an ontology, and domain experts who want to query the ontology [KJM+12]. Knowledge engineers need to classify the whole ontology regularly in order to check for inconsistencies, and to make sure that the inferred subsumptions are as intended [MDOS14]. This can pose a severe bottleneck during development, even if classification could be done in less than five minutes, which it very often cannot [GMPS13]. The problem with

[1] Gives all the answers (complete) and only the right ones (sound) and is guaranteed to finish at some point (terminating).

languages based on expressive Description Logic such as OWL is a high worst case complexity (for reasoning problems) and a complex input space. This makes research into OWL reasoning one of the primary concerns of the OWL/Description Logic community.

Reasoning with Description Logic/OWL ontologies in general and classification in particular is intractable [Kaz08, HKS06, Tob01]. Before classification was shown to behave quite reasonably in practice [Hor97], the daunting worst case complexity (2NExptime for $\mathcal{SROIQ}$ [Kaz08]) of the satisfiability problem (a basic reasoning problem underlying others, such as classification), fostered the wide-spread belief that sound and complete reasoning was impractical. Regardless of that, decision procedures were implemented for more than two decades before OWL (and the first OWL reasoner) was developed. In order to deal with the high complexity, the community came up with ever more sophisticated reasoner optimisations. The late 1990s saw the first highly optimised reasoners for expressive Description Logics [Hor98, PS00, HM01].

Over the years many efficient reasoners were developed[2] to handle expressive OWL ontologies, offering services such as classification, instance checking or query answering, in many cases with acceptable performance [GMPS13]. Even larger ontologies such as SNOMED have recently been classified in less than 5 seconds[3] [KKS14], and a wide range of ontologies can now be classified in less than three minutes by novel, highly optimised reasoning systems [PMG+15, SLG14]. As we will see in this thesis however, some complex, and often large, ontologies still constitute a challenge. Apart from that, it is quite possible that current ontologies are deliberately kept small or inexpressive in order to be processable by reasoners and that ontologies in the future are much more complex, and larger than the ones we currently have to deal with. Therefore, the quest for better reasoner optimisations is ongoing, and remains to be a core area of Description Logics research.

Our current understanding of reasoner performance for OWL ontologies is limited. For more than two decades, the community has obtained important complexity results, focusing however on the worst case runtime. Worst case runtime is only of limited interest to understanding reasoning with real problems. For example, it is possible that no real problem triggers the exponential

[2] `http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/`

[3] Intel Core i7-2630QM 2GHz quadcore CPU, 6 GB RAM, Windows 7.

behaviour of the, theoretically, intractable reasoning procedure. The matter is made worse by the amount of, frequently intertwined, optimisations developed for reasoners, and the different calculi used to determine satisfiability and therefore subsumption. For example, depending on the fragment of Description Logic at hand,[4] (mostly incomplete) structural approaches were used in the 1980ies, tableau-based approaches started to be used in the 1990ies and hyper-tableau and consequence based approaches after 2000 [BCM+07]. As of now, we cannot confidently answer the question what makes reasoning hard, an important pre-requisite for example for directing the quest of optimisation development.

1.2 The Problem of Modular Reasoning

We can distinguish optimisations for classification by those targeted at

- avoiding subsumption tests when traversing the space of possible subsumptions (traversal space),
- making necessary tests cheaper by optimising the calculus,
- replacing subsumption tests fully or partially by cheaper alternatives and
- those that rewrite the input into an easier logically equivalent one [BCM+07].

One branch of optimisation development is the exploitation of logical modules for classification, creating optimisations primarily targeted at avoiding tests and making them easier. Modules are subsets of an ontology that come with certain guarantees for reasoning [GHKS08]. The central rationale is that reasoning with a module of an ontology should be *easier than reasoning with the whole*. This "easyfication" could be for example induced by avoiding a large number of negative subsumption tests or by making individual tests easier (because reasoning in a small, self contained subset of the ontology is easier).

The first optimisation that exploited modularity, by now a standard aspect of many state-of-the-art reasoners, is incremental reasoning, which uses modules to isolate the part of the ontology that needs to be re-classified in the face of change [GHWK07]. In recent years, module extraction has been used to decompose the entire ontology into a set of possibly overlapping modules that could be reasoned with separately [RGH12, TP12a]. The hope was that reasoning with these parts was overall easier than reasoning with the whole. Modularity-based

[4]The reader may think of a fragment as a grammar for the ontology language, allowing for a fixed set of constructors to create statements of knowledge.

classification is motivated by the following three assumptions:

1. One or more modules in the decomposition can be processed by a more efficient delegate reasoner than the one that would be required to process the whole.
2. The traversal space for subsumption tests *shrinks* significantly.
3. Subsumption testing gets easier because there are fewer irrelevant axioms.[5]

While these assumptions appear reasonable, virtually nothing is known about the extent to which modular decompositions have the desired beneficial effects on classification performance. Some preliminary experimentation with MORe[6] and Chainsaw[7] suggested that there were some, occasionally significant, benefits, as well as a number of cases were modularity had a significant detrimental effect [RGH12, GBJR+13, PMG+15]. This makes a systematic investigation into the modular reasoning techniques worthwhile.

Other than that, reasoning with modular decompositions has hardly received any mainstream attention. The *central arguments against* employing modules are the occasionally severe (and often decisive) overhead in computing them, the overlap between the modules of decompositions in practice, and the uncertainty that the consequences of pruning the traversal space and "easifying" subsumption tests is any more than marginal. While "true" module extraction is as hard as reasoning, we make use of syntactic approximations [GHKS08]. Syntactic locality-based module extraction for example can be done in polynomial time (lower worst case complexity than reasoning), which however does not mean that it is "easy" in practice. Overall, there is a justifiable atmosphere of doubt whether using modular decompositions for classification is, despite being well motivated and often beneficial in practice, generally worth it.

[5] In this thesis, we are ignoring any benefits that might come due to the potential of classifying modules in the decomposition in parallel.

[6] MORe is a reasoner that exploits the cheaper delegate assumption by decomposing an ontology into two modules, one of which (hopefully large) can be processed by a very efficient delegate reasoner.

[7] Chainsaw is a reasoner that decomposes the ontology into hopefully very small, but potentially overlapping modules.

1.3 The Problem of Understanding Reasoning Performance

Current attempts at black box modular reasoning have yielded mixed results with little understanding of what works, and why, and whether deeper modifications of reasoners are necessary to realise systematic gains. Worst case complexity studies are limited in the sense that an algorithm may only be intractable in odd corner cases that do not occur in practice. Therefore, it is desirable to study how reasoning algorithms fare for realistic inputs. A first thought would be to conduct average-case complexity studies. These studies are still formal, i.e. they do not require observing the performance of particular implementations. One of the problems with average-case complexity studies is that they require us to define a problem distribution that we believe to be somehow close to the natural distribution. DL concept expressions can be very complicated, and there is very little knowledge in the literature about how ontologies in the wild are shaped. Efforts for generating random reasoning problems generally rely on fixing a large number of parameters [Hla05], which may or may not resemble real problems. To our knowledge, there are no average-case complexity studies conducted for expressive Description Logics, and we believe the problem lies with the difficulty of defining a suitable problem distribution. Recently, some advances have been made on understanding the hardness of reasoning using parametrised complexity studies. Simançik et al. [SMH14] explore the measure of a graph's similarity to a tree (tree-width) as a central metric to quantify the hardness of subsumption reasoning in $\mathcal{ALCI}$. The authors conduct the first study on fixed-parameter tractability for Description Logic reasoning, showing that tree-width can be taken as an indicator for the hardness of sets of propositional clauses. While these are certainly important to understand the hardness of satisfiability testing (albeit in a setting of less expressivity than OWL), they do not cover the complexity for example of classification algorithms, which rely (among others) on traversal algorithms in addition to satisfiability checking. We can view the traversal algorithm as a super-process aiming at reducing the space of necessary subsumption tests, and subumption tests as sub-processes that involve the reasoner to either call the satisfiability engine or conducting a cheaper alternative test. Because of the limitations of formal approaches, the dawn of highly optimised systems for expressive Description Logics towards the end of the 1990ies

such as FaCT [Hor98], Racer [HM01] and DLP [PS00] also came with a paradigm shift from purely formal towards empirical investigations in the DL community. In order to understand the hardness of reasoning, empirical studies of reasoning performance on real[8] ontologies became indispensable.

Empirical studies on reasoning performance come in a variety of shapes. One branch of investigations explores the use of Machine Learning techniques to expose ontology features that are somehow relevant to predict reasoner performance [KLK12, SSB14]. These approaches claim to work quite well, but while certainly providing food for thought, they do *not* provide us with models that are explanatory[9] and tend to be evaluated against corpora with a large proportion of easy ontologies, failing in particular on the very hard - and thus interesting - cases. For example, it has been shown that most ontologies sampled from BioPortal or other corpora are comparatively easy for most modern reasoners [GMPS13]. A more recent work by Alaya et al. [AYL15] predicts reasoner robustness based on a set of structural features, unveiling key features of an ontology that can explain robustness, such as axiom depth and the values of number restrictions in axioms, again using supervised learning techniques. Learning-based approaches are strongly backward looking. New ontologies can bring new challenges, and may not fit well into such predictive models.

One of the contributions of this thesis is concerned with *reasoner benchmarks*. Reasoner benchmarking is the process of testing the implementation of reasoning tasks in reasoning systems in order to determine a set of performance characteristics. While this is usually understood to be the measurement of execution time, it is not limited to that. Some benchmarks have been developed to measure power or memory consumption [VNP15], some with the intention to capture the performance of traversal algorithms by measuring how well they avoid tests [GHM+12]. Benchmarks for classification algorithms that are concerned with execution time usually focus on overall classification time. Little to no interest is paid for example on the timings of individual subsumption tests triggered during classification. In practice, most benchmarks are implemented in an ad-hoc fashion and are not easily re-usable.

The problem of *modular reasoning* lends itself very well to develop such benchmarks for understanding reasoner performance because it abstracts away from the

[8] rather than artificially generated

[9] "These n features can be linearly combined to predict reasoning performance with x accuracy"

complexity of the delegate reasoner's[10] internals, and focuses on questions that can be tested in controlled experiments without too much confounding by the complex interplay of potentially conflicting optimisations within the (delegate) reasoner. For example, we want to check the assumptions that reasoning with a module is actually easier than reasoning with the whole ontology, or test whether the hardness of individual subsumption tests drops on average when triggered within a module compared to when triggered in the context of the whole ontology.

In order to determine whether and why the use of modules for classification is generally worth it, a better understanding of reasoning behaviour under modularity is needed. For that, more fine grained benchmarks and more systematic, independent empirical investigations are being designed and conducted by our community [PMG+15]. In this thesis, understanding OWL classification as such is not the focus. In particular, we do not aim to understand why current reasoners behave the way they do, and we do not try to understand why particular ontologies trigger a particular reasoner behaviour. Our focus lies on developing tools and methodologies to understanding how modularity, and in particular module-based decompositions, impact classification performance—clearly isolating positive and negative effects on performance.

1.4 Contributions

The first contribution of this thesis is the systematic, empirical exploration of the potential of modularity to improve classification time by reducing the average test hardness and/or reducing the number of tests. In particular, we will attempt to dispel or confirm some of the doubts that are currently based mostly on intuition and folk-wisdom:

- We prove that modules are occasionally harder than the whole ontology.
- We prove that modularity often has a positive effect on the hardness of subsumption tests, but, depending on the ontology and the delegate reasoner, not always a significant one.
- We prove that the question of using module-based decompositions for OWL classification for pruning the subsumption test space has an occasionally

[10] The "actual" reasoner that computes a partial answer to the reasoning request based on a given module.

highly beneficial, occasionally highly detrimental effect—depending on the ontology.

The second contribution of this thesis is to further the methodological foundations of experimenting with reasoners and modules through novel experiment designs, APIs and large well characterised datasets:

- We have designed a framework that supports experimenting with reasoners capturing the various stages a reasoner goes through during classification, including individual subsumption tests, and show how this enables new insights into reasoning performance (Reasoner Stage Benchmark).
- We have developed an extensible framework for analysing modularity-based classification techniques (Katana), and have implemented 5 approaches, 3 of which have not yet been considered.
- We have created, published and analysed a large corpus of fine grained observations on classification, including timings of individual subsumption tests triggered during classification.
- We present some of our efforts involving ontology corpora, most importantly assembly and in-depth characterisation.

Note that we are *not* exploring the benefits of integrating efficient delegate reasoners, the main rationale underlying the MORe reasoner. Our frameworks to allow the integration of specialised delegates however, and characterising their effect will be part of future work.

1.5 The Story

The primary goal of this thesis is to determine the viability of modular decompositions to improve classification performance. We will briefly describe the basics of Description Logic reasoning and modularity in Chapter 2. Chapter 3 will describe in detail the idea of modularity-based classification, in particular with respect to associated threats and potential benefits. We will cover the existing implementations, and explain our own model of modular reasoning. In Chapter 4 we will describe our experimental framework, introducing the Reasoner Stage Benchmark, Katana and the OWL Experiment API and discussing the reasoners and datasets used in our experiments.

In the following three chapters, we will address some of the main questions

with respect to modularity-based classification. In Chapter 5 we explore the fundamental conjecture that a module is never harder than its parent ontology. We show empirically that *pathological modules* exist and the conjecture does not hold, and proceed to explore the prevalence and harmfulness of pathological modules as they are used by state-of-the-art modular reasoning approaches. In Chapter 6 we investigate how modularity effects subsumption test hardness, first isolating ontologies that can potentially benefit from the effect and then determining the potential magnitude of the effect. This chapter serves both as an in-depth characterisation of the phenomenon of test hardness under modularity and builds on the observations in Chapter 5 by revealing that, while individual tests are on average easier under modularity, most test do not change in hardness at all. Finally, Chapter 7 investigates modularity-based classification approaches in action. We introduce three untested approaches, and compare them, along with variants of the Chainsaw and MORe approaches, with respect to overall performance, computational overhead, redundancy, traversal space and subsumption test hardness.

1.6 Published Work

Some of the work we present as part of this thesis has been presented at international conferences and workshops.

- Matentzoglu, N., Parsia, B., Sattler, U. An Empirical Investigation of Difficulty of Subsets of Description Logic Ontologies, International Workshop on Description Logics 2014 [MPS14] (early version of Chapter 5)
- Matentzoglu, N., Sattler, U., Parsia, B. Empirical Investigation of Subsumption Test Hardness in Description Logic Classification, International Workshop on Description Logics 2015 [MSP15] (early version of Chapter 6)
- Matentzoglu, N., Bail, S., Parsia, B. A Snapshot of the OWL Web, ISWC 2013 [MBP13b] (foundational work on ontology corpora)
- Matentzoglu, N., Bail, S., Parsia, B. A Corpus of OWL DL Ontologies, International Workshop on Description Logics 2013 [MBP13a] (foundational work on ontology corpora)
- Matentzoglu, N., Leo, J., Hudhra, V., Sattler, U., Parsia, B. A Survey of Current, Stand-alone OWL Reasoners, International Workshop on OWL Reasoner Evaluation (ORE) 2015 [MLH+15] (reasoner survey presented as

part of Chapter 4)

- Matentzoglu, N., Parsia, B. OWL/ZIP: Distributing Large and Modular Ontologies, International Workshop on OWL: Experiences and Directions (OWLED) 2014 [MP14c] (technique used for distributing decomposed ontologies, Chapter 4)
- Matentzoglu, N., Parsia, B. The OWL Full/DL Gap in the Field, International Workshop on OWL: Experiences and Directions (OWLED) 2014 [MP14b] (technique used for OWL Full repair during corpus pre-processing, Chapter 4)
- Matentzoglu, N., Tang, D., Parsia, B., Sattler, U. The Manchester OWL Repository: System Description, ISWC 2014 Posters and Demonstrations [MP14a] (repository we developed to share datasets for experimentation)
- Gonalves, R.S., Matentzoglu, N., Parsia, B., Sattler, U. The Empirical Robustness of Description Logic Classification, ISWC 2013 Posters and Demonstrations [GMPS13] (early benchmark, pre-courser of thesis benchmark)
- Gonalves, R.S., Bail, S., Jimnez-Ruiz, E., Matentzoglu, N., Parsia, B., Glimm, B., Kazakov, Y. OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report, International Workshop on OWL Reasoner Evaluation (ORE) 2013 [GBJR+13] (ORE Reasoner competition, sampling methodology, data set curation)
- Lee, M., Matentzoglu, N., Sattler, U., Parsia, B. A Multi-reasoner, Justification-Based Approach to Reasoner Correctness, ISWC 2015 [LMPS15] (Methodology used to establish reasoner correctness)

Chapter 2

Background

In this chapter we cover some of the basic notions of reasoning and modularity. We will briefly discuss the architecture of OWL reasoners and some of the key reasoning tasks and describe the reasoning task of classification in more detail with respect to two important calculi. We will conclude the section on reasoning with a discussion of reasoner optimisations relevant to OWL classification. In the modularity section we introduce syntactic $\bot$-modules and their properties relevant to module-based classification. We will conclude the chapter with a definition of classification-preserving decompositions and some examples. The main goals of this chapter are:

- Introducing the notation we use throughout this thesis.
- Setting the scene for module-based classification of OWL ontologies by introducing Description Logic classification.
- Introducing our notion of classification-preserving decomposition.
- Motivating our choice of syntactic locality-based $\bot$-modules to underpin decompositions for classification.

2.1 Terminology Used

In Table 2.1, we introduce some of the basic terminology we will use throughout this thesis.

Ontologies (see definition in Table 2.1) can be dependent on other ontologies, most notably through the `owl:imports` relation. It is important to note that we treat the *entire imports closure* as the ontology, not merely the *root importing ontology*, see also Bail et al. [BPS11].

The reader will notice a potentially confusing mixture of the OWL and DL terminology, depending on the context of the paragraph. For example in the context of OWL, we refer to class expressions and classes, while the DL community refers to the same ideas as concepts and concept names. Another example is the notion

Table 2.1: Important terms used throughout the thesis.

Term	Denoted by	Definition
Axiom	none	A statement in OWL, for example $A \sqsubseteq B$, Functional(R).
Logical axiom	α	Subset of axioms that excludes annotation assertions and entity declarations.
Imports closure	none	The set of ontologies directly or indirectly imported by a given root ontology.
OWL ontology	$\mathcal{O}$	The set of logical axioms in the imports closure of an ontology.
Signature	$\widetilde{\mathcal{O}}$	The set of all names occurring in the ontology, including class names, property names and individuals.
Classes in Signature	$\widetilde{\mathcal{O}}_C$	The set of all class names in the ontology.
ABox	none	The set of all class assertions and property assertions in the ontology (A for assertional).
TBox	none	The set of all logical axioms in the ontology excluding ABox axioms (T for terminological).

of "role" in DL, which can either refer to an object property or a data property in OWL. We try to use the terminology that we find appropriate to the given context, but use both terminologies in a way that is completely interchangeable.

2.2 The Web Ontology Language and its DL Underpinnings

The Web Ontology language (OWL) is a family of knowledge representation languages used for expressing ontologies. OWL allows us to talk about entities such as concepts of a domain, instances of concepts and roles connecting instances (and potentially concepts and data values) using a pre-defined vocabulary. OWL 2 [GHM+08], the successor of OWL, is comprised of *two main species* of different expressivities: OWL 2 DL and OWL 2 Full. As we have mentioned before, the underlying formalism of OWL 2 DL is the Description Logic $\mathcal{SROIQ}(D)$ [HKS06]. While OWL 2 DL is defined by Description Logic semantics (*Direct*

Semantics), OWL 2 Full has an RDF-based semantics[1], which is a superset of the OWL 2 Direct Semantics. OWL 2 Full does not impose any restriction on the usage of the language constructs of OWL.

OWL reasoners, however, are usually restricted to ontologies in (a subset of) OWL 2 DL. We generally regard OWL 2 DL as a syntactic variant of $\mathcal{SROIQ}(\mathcal{D})$, that has a few additional features such as metamodelling[2], annotations and keys.[3] There are three specified *profiles* of OWL 2 DL, namely OWL 2 EL, OWL 2 QL and OWL 2 RL, for which the community has developed decision procedures of only polynomial complexity (reasoning with the entirety of OWL 2 DL is of exponential complexity). All three are tailored to particular use cases. OWL 2 QL can serve as basis for efficient query answering, often in conjunction with database technologies. OWL 2 RL is used if the goal is to capture rule-like (if-then) knowledge and enable the interaction with rule engines and rule extended database management systems, and then provide efficient reasoning (over potentially large datasets) in a scalable way [MGH+12]. OWL 2 EL is supposed to be used for modelling large ontologies (mainly the schema or TBox part of them), allowing for very efficient classification. All three profiles intersect in terms of expressivity, i.e. it is possible for any given ontology to fall under all three profiles at once. A notable exception to reasoners being restricted to OWL DL is the work by Schneider et al. [SS11], who have translated a large fragment of the OWL 2 Full semantics into First Order Logic, and explore the use of First Order Logic theorem proving systems for reasoning.

OWL can be represented by a wide range of *syntaxes*, some of the most widely used being RDF/XML, OWL/XML, Functional Syntax and Manchester Syntax, the latter being usually associated with better readability by humans. For a full definition of OWL 2, we refer the reader to the official specification.[4]

2.3 Brief Overview of Reasoning in OWL

An OWL reasoner $\mathcal{R}$ is a system that solves OWL reasoning problems, making implicit knowledge in the ontology explicit. Figure 2.1 provides a high-level schematic of the architecture of a reasoner. A reasoner offers a range of logical

[1] `http://www.w3.org/TR/owl2-rdf-based-semantics/`
[2] Also called punning, see `http://www.w3.org/TR/owl2-syntax/#Metamodeling`
[3] `http://www.w3.org/TR/owl2-syntax/#Keys`
[4] `http://www.w3.org/TR/owl2-syntax/`

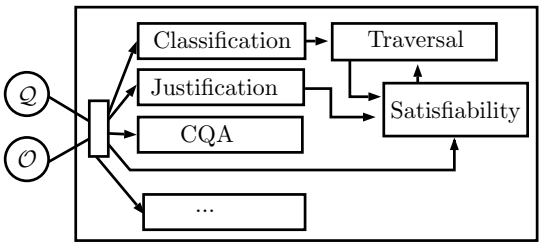


Figure 2.1: A reasoner offers a range of logical services such as classification, justification generation and conjunctive query answering (CQA). Input to these services are an ontology $\mathcal{O}$, and a query $\mathcal{Q}$ (is satisfiable? is consistent?, classification?). Services are sometimes built on other services, such as classification on satisfiability.

services, for example consistency checking, classification, justification generation and conjunctive query answering (CQA). Input to these services are usually an ontology $\mathcal{O}$ in a language (a fragment of) OWL, possibly extended[5], and a query $\mathcal{Q}$, which ranges from decision problems ("Is the ontology consistent") to listing problems ("Give me all the instances of C"). Reasoning services are implemented using one or more *calculi*, most notably those based on (hyper-)tableau and those based on saturation (consequence-based reasoning techniques). Some reasoning services can make use of other ones: For example, while performing classification, a tableau-based reasoner typically employs a traversal strategy to iterate the pairs of concept names that need to be checked for subsumption. Unless the test can be somehow avoided (through one of many potential optimisations), the reasoner then calls the service for subsumption testing, which in turn decides whether the subsumption holds by querying the *satisfiability engine*. For example, in order to determine whether a concept A is subsumed by a concept B, the reasoner tests the satisfiability of the concept $A \sqcap \not B$. Upon determining satisfiability (or non-satisfiability), the traversal algorithm progresses to the next pair of concept names. The answer to a query can be a Boolean answer (consistent), a fully classified ontology, the set all members of a class – depending on the nature of the query and the particular service invoked.

The general motivation for using formalisms based on Description Logic semantics is the availability of *sound*, *complete* and *terminating* reasoning procedures, i.e. decision procedures. Sound reasoning procedures do not give wrong

[5]Measurement units, probability and fuzzy logic are examples of existing extensions.

answers, complete ones give all the right answers, and terminating ones are guaranteed to finish in finite time. The reasoning procedures used for Description Logic reasoning are all sound, complete and terminating with respect to a fragment of Description Logics. However, (1) the same does not necessarily hold for the implemented systems and (2) a reasoning procedure complete for one fragment is not necessarily complete for another.

All Description Logic reasoners (to our knowledge) aim to provide *sound* and *terminating* reasoning services for some language, and with a few exceptions, *complete* ones. However, it is quite unlikely that a given OWL reasoner is completely bug-free [LMPS15]. An unsound or non-terminating algorithm for OWL reasoning is almost certainly the consequence of a bug. *Incompleteness* however can be either *intended* (approximate reasoning) [TPR10], *accidental* (bugs in the implementation) [LMPS15] or simply a consequence of incomplete *coverage*. For example, while the features of OWL that are important for reasoning (possibly with the exception of punning and keys) are covered by the Description Logic language $\mathcal{SROIQ}(\mathcal{D})$ [HKS06], implementations do *not always cover all of them*. Reasoners are more often than not a *work-in-progress*, and even if the underlying procedures are sound and complete, the reasoner may not (yet) fully implement all OWL features. This usually takes the form of incomplete datatype coverage, which has been for example a problem for the reasoner FaCT++ [TH06] in reasoner competitions, which covers only a subset of the OWL 2 datatype map. This is an important aspect that needs to be taken into account when *comparing* the performance of two reasoners: they do not always make the effort needed to be strictly *complete*. Another example of *incomplete coverage* is the treatment of ELK [KKS14] as "mostly" complete with respect to OWL 2 EL ontologies.[6] These sources of incompleteness should be kept in mind when interpreting benchmarking results.

2.3.1 Reasoning Tasks for Description Logic Ontologies

OWL reasoners provide many standard and non-standard *reasoning services*. A reasoning service implements a *DL reasoning task*. The standard Description Logic reasoning tasks are classification, entailment checking, consistency checking, conjunctive query answering and realisation [BCM+07]. Among the non-standard

[6] ELK does not yet fully support datatypes (like lexical-value mappings), or imposes restrictions on the position where certain constructors like `ObjectComplementOf` are allowed.

reasoning services are explanation generation [Hor11], axiom pinpointing [BP10] and (semantic) module extraction [GHKS08]. Reasoning tasks can be roughly grouped by their applicability to either the ABox or the TBox of an ontology. The most notable categories are those reasoners whose primary task is the efficient classification of the TBox and those that answer queries over the ABox, mainly conjunctive query answering [MLH+15].

In this thesis, we are concerned with *TBox reasoning tasks*, most specifically *classification*. Generally speaking, classification is the process that takes as an input an ontology $\mathcal{O}$ and decides, for all names $A, B \in \widetilde{\mathcal{O}}_C$ whether $A \sqsubseteq B$ (read: A is a sub-class of B). Unlike the reasoning *task*, the reasoning *service* of classification (as implemented by a reasoner) is *not uniformly defined* in terms of scope and output. What *names* are considered and what is returned varies across implementations. Most typically, we consider the classification of classes. However, some reasoners may do some additional, not strictly necessary work, like classifying object properties, which can make the interpretation of timings more challenging. We usually say that the output of classification is the *inferred class hierarchy*. In practice, this can take one of two, and possibly more, forms: (1) The reasoner keeps an internal representation of the classified hierarchy, and returns a Boolean[7] to indicate that it is now ready to answer queries over the class hierarchy. (2) The reasoner returns the class hierarchy[8] encoded as a set of OWL axioms. Implementations also vary in the inclusion of `OWL Thing` and `OWL Nothing` in the class hierarchy as well as the representation of equivalent classes and unsatisfiable classes. In the practice of reasoner benchmarking, inspecting the inferred class hierarchy (or comparing them between reasoners) usually involves normalising the output of reasoners, for example by manually constructing the hierarchy by querying the reasoner iteratively for all sub- and superclasses of all class names $A \in \widetilde{\mathcal{O}}_C$.

2.3.2 OWL Ontology Classification

There are two fundamentally different main approaches to classify OWL ontologies: *consequence-based* classification and classification based on traversal

[7] Or nothing at all in the case of the OWL API [HB11], the de-facto standard for manipulating ontologies and using reasoners in Java-based systems.

[8] Hasse Diagram of the partial order induced by $\sqsubseteq$.

and subsumption testing, most importantly those based on tableau and hypertableau. Tableau-based approaches attempt to construct a class graph by conducting a series of subsumption tests. Each subsumption test involves building a *counter-model*; if the construction fails, the subsumption holds, and vice versa. Consequence-based techniques on the other hand determine subsumptions through successive applications of *derivation rules*. Each rule defines a conclusion, a set of premises and a side condition. Both approaches differ not only methodologically, but also in terms of *applicability*. These differences will be discussed in Section 2.3.2.

Tableau-based classification algorithms have two main ingredients: a *traversal algorithm* that orders the tests necessary to classify an ontology, and a *satisfiability engine* that is responsible for determining concept satisfiability (for example by building the counter-models). A naive traversal algorithm would iterate through all ordered pairs of concept names $A, B \in \widetilde{\mathcal{O}}_C$ and determine whether $\mathcal{O} \models A \sqsubseteq B$, requiring a total of n^2 tests, where $n = |\widetilde{\mathcal{O}}_C|$. This is clearly inefficient, given the tree like structure of a typical class hierarchy. For example, if we find that $\mathcal{O} \not\models A \sqsubseteq B$ then we do not need to check whether $\mathcal{O} \models A \sqsubseteq D$ for any D with $\mathcal{O} \models D \sqsubseteq B$, see Figure 2.2. Therefore, folklore suggests to assume a tighter upper bound of $n * log(n)$ tests, which reflects the complexity of traversing a tree. While this is (as we will see in Chapter 6) much closer to the empirical upper bound than n^2, it is occasionally exceeded. One trivial example of an ontology that is not tree-shaped is one with all axioms having mutually disjoint signatures. A naive traversal algorithm would have to conduct all n^2 tests in that case. For that reason, we will consider both whenever we are referring to upper bounds in our discussions. Examples of tableau based reasoners are Pellet and FaCT++. Note that there are no up-to-date numbers on how the class graph of an OWL ontology is typically shaped. A fairly old survey from 2006 classifies a corpus of ontologies according to a taxonomy of class graph shapes [WPH06]. It is suggested that there are quite a number of class graphs that are not merely trees, but often multi-trees or even directed acyclic graphs. It is likely (but not known strictly speaking) that this is similarly true for the ontology landscape today.

In contrast, *consequence-based* reasoners classify an ontology by systematically applying derivation rules to the axioms in $\mathcal{O}$. ELK, a well known consequence-based reasoner, implements classification (in a nutshell) as follows. All axioms

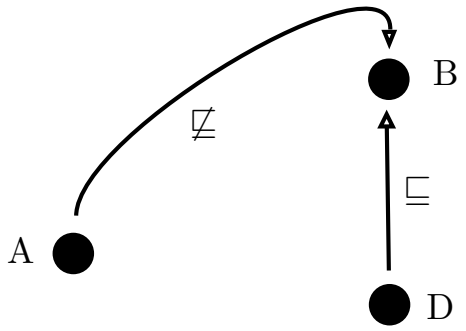


Figure 2.2: If A is not a subclass of B, than it can also not be a subclass of any of B's subclasses.

are added first to a todo-list. Then they are, one by one, added to a *closure*, after which a number of derivation rules are checked for applicability. The consequences of applied rules are added to the todo list. Once the todo-list is empty, the classification is complete.

In order to *reduce the number of tests*, numerous optimisations were developed for *tableau-based classification algorithms*, most notably the family of *Enhanced Traversal* algorithms. The overarching rationale is that the predominantly tree-like structure of the class hierarchy allows for reducing the search space for subsumptions when gradually constructing the class hierarchy. We will discuss an example of an Enhanced Traversal algorithm in Section 2.3.2. While consequence-based techniques do not consider non-subsumptions, and therefore do not need traversal algorithms in the same sense as tableau-based techniques, they are not completely immune to redundancy: each conclusion can be potentially derived multiple times [KKS14].

Each individual subsumption is, depending on the underlying calculus, determined by exhaustively applying a number of derivation rules, by trying to construct a model of the concept and the complement of its potential subsumer or by a cheap alternative test like a query to the transitive closure of the class graph. This task can be *very hard*: Especially model construction for expressive ontologies can have a complexity up to N2ExpTime in the worst case. While the quadratic worst case complexity of traversal (n^2) appears tiny compared to the worst case complexity of subsumption testing, it is not always certain *which of the two effects the hardness of a given classification more* in practice. For example, Glimm et al. [GHM$^+$12] have shown that their *Novel Approach*, an optimised traversal algorithm for ontology classification, can boost performance sometimes

by more than an order of magnitude - merely due to *test avoidance.* It is clear that any serious attempts at optimising classification need to address both avoidance and hardness, regardless of their worst case complexity upper bounds.

Classifying a reasoner as either consequence-based or (hyper-)tableau-based is rarely straight forward. At least the general purpose reasoners that cover the entirety of OWL 2 DL often employ *hybrid techniques*, for example combining saturation-based or consequence-based techniques with tableau-style techniques, or are based on *modular techniques.* In terms of hybrid techniques, we can distinguish between *quasi-hybrid* reasoners and *true hybrids.* Quasi-hybrid reasoners usually employ some cheap up-front case distinction to determine whether an ontology should be processed by a cheaper, usually consequence-based engine, or a model-construction-based technique for higher levels of expressivity. For example, Pellet uses its internal $\mathcal{EL}$-classifier for ontologies in (some variant of) $\mathcal{EL}$. This "optimisation" is becoming standard practice, due to the large potential performance gains and the cheapness of the upfront expressivity check. True hybrid reasoners like Konclude [SLG14] come with a very tight integration of the different calculi, for example combining saturation-based techniques with tableau algorithms [SGL14]. *Modularity-based* reasoners such as Chainsaw and MORe, sometimes referred to as "portfolio" reasoners, should also not be classified into one of the three main categories. Their main purpose is to compose the ontology into modules and then feed the individual bits to delegate reasoners.

In the following, we will exemplify the classification algorithms for both approaches with respect to a simple ontology $\mathcal{O}_{Ex}$ with two axioms:

- $A \sqsubseteq B$
- $B \sqsubseteq C$

Traversal / Tableau-based classification

A reasoner implementing a tableau-based procedure starts classification by initialising the target class graph with $\bot \sqsubseteq \top$[9] (tautology, i.e. always true). Next, the classification algorithm iterates through the class names in $\widetilde{\mathcal{O}_{ExC}}$. Classification is non-deterministic, e.g. the order in which concepts are classified is not defined. For now, we assume lexical order and classify the classes A, B and C in that order. The classification algorithm has a top-down and a bottom-up stage.

[9] $\top$ is the top concept, and includes all elements in a domain, and $\bot$ is the bottom concept, which represents the empty set.

At first we check whether $\mathcal{O}_{Ex} \models A \sqsubseteq \top$. That trivially holds, so we move on to check whether $\mathcal{O}_{Ex} \models A \sqsubseteq \bot$, or, in other words, whether A is unsatisfiable. A is unsatisfiable if there is no model of $\mathcal{O}_{Ex}$ in which A can have an instance. This test is implemented as a call to the satisfiability engine, which tests the satisfiability of the concept $A \sqcap \neg\bot$. Since $\neg\bot$ is equal to $\top$, we merely test the satisfiability of A. Since A is satisfiable in our example, the subsumption *does not hold*. Now the bottom up stage begins. We start by checking whether $\mathcal{O}_{Ex} \models \bot \sqsubseteq A$. Since $\bot$ is a subclass of everything, this subsumption holds by default. Next we test whether $\mathcal{O}_{Ex} \models \top \sqsubseteq A$. Since $\top \sqcap \neg A$ is satisfiable, the subsumption does not hold. As a consequence, we insert A in the class graph between $\top$ and $\bot$. Next we move on to class B. For readability, we will present the remaining steps in the form of a list.

1. B
   (a) Top-down
      i. $\mathcal{O}_{Ex} \models B \sqsubseteq \top$ holds trivially
      ii. $\mathcal{O}_{Ex} \models B \sqsubseteq A$ does not hold because $B \sqcap \neg A$ is satisfiable
      iii. $\mathcal{O}_{Ex} \models B \sqsubseteq \bot$ does not hold because B is satisfiable
   (b) Bottom-up
      i. $\mathcal{O}_{Ex} \models \bot \sqsubseteq B$ holds trivially
      ii. $\mathcal{O}_{Ex} \models A \sqsubseteq B$ holds because $A \sqcap \neg B$ is *not* satisfiable
      iii. $\mathcal{O}_{Ex} \models \top \sqsubseteq B$ does not hold because $\top \sqcap \neg B$ is satisfiable
   (c) Inserting B between $\top$ and A.
2. C (compare also Figure 2.3).
   (a) Top-down
      i. $\mathcal{O}_{Ex} \models C \sqsubseteq \top$ holds trivially (Step 1 in figure)
      ii. $\mathcal{O}_{Ex} \models C \sqsubseteq B$ does not hold because $C \sqcap \neg A$ is satisfiable (Step 2 in figure)
      iii. $\mathcal{O}_{Ex} \models C \sqsubseteq A$ can be avoided because $\mathcal{O}_{Ex} \not\models C \sqsubseteq B$
      iv. $\mathcal{O}_{Ex} \models C \sqsubseteq \bot$ does not hold because C is satisfiable (Step 3 in figure)
   (b) Bottom-up
      i. $\mathcal{O}_{Ex} \models \bot \sqsubseteq C$ hold trivially (Step 1 in figure)
      ii. $\mathcal{O}_{Ex} \models A \sqsubseteq C$ holds because $A \sqcap \neg C$ is *not* satisfiable (Step 2 in figure)
      iii. $\mathcal{O}_{Ex} \models B \sqsubseteq C$ holds because $A \sqcap \neg C$ is *not* satisfiable (Step 3 in

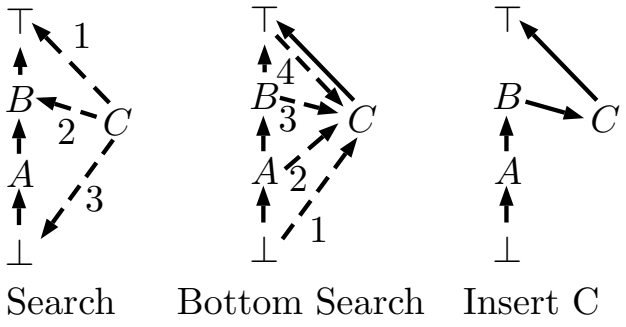


Figure 2.3: Illustration of the top and bottom search phase for the example in the text.

figure)

iv. $\mathcal{O}_{Ex} \models \top \sqsubseteq C$ does not hold because $\top \sqcap \neg C$ is satisfiable (Step 4 in figure)

3. Inserting C between $\top$ and B.

After that, the classification of $\mathcal{O}_{Ex}$ is completed. Note that tableau-based classification algorithms have two major sources of non-determinism:

- The order of the subsumption tests (traversal space).
- The order of branch explorations during model construction.

Both sources of non-determinism can have severe implications for performance, and need to be taken into account when trying to understand classification performance. Note that for Horn-$\mathcal{SHIQ}$, a Description Logic language that contains all three OWL 2 profiles, we have deterministic reasoning procedures. However, reasoners such as FaCT++, Pellet and JFact do not implement these procedures, so they can still introduce non-deterministic choices.

Consequence-based classification

Now we will look at the consequence-based equivalent, as for example employed by ELK. For brevity, we will omit the $R_\top$ rule, which is used to derive that all concepts are sub-concepts of $\top$. In contrast to tableau-based classification, we do not classify concepts one-by-one. Instead, we loop through the axioms in $\mathcal{O}$ and apply a number of derivation rules until no further rule can be applied. This process is called *saturation*, and the fully classified ontology is called *saturated*. A basic implementation works as follows: we start with an empty closure (the place were we will collect the expressions between which all derivation rules have been fully applied), and a todo list consisting of the axioms in the ontology. The

first rule we apply is the R_0 rule, which states that all classes are subclasses of themselves. We start the saturation by applying the R_0 rule to all class names appearing on the left hand side of an axiom. A and B both appear on the left hand side of axioms, so we add $A \sqsubseteq A$ and $B \sqsubseteq B$ to the closure. All subsequent rule applications are executed as follows: For each axiom in the todo list, we:

1. Add axiom to closure.
2. If axiom was already in closure move on to next axiom.
3. Else apply inference rules one-by-one by treating axioms already in the closure as premises and the axiom at hand as the side condition.
4. If rule applies, draw conclusion.
5. If conclusion is already in closure, do nothing.
6. Else add it to todo list (not immediately to the closure!).

With respect to our example, we take the first axiom in our todo list, $A \sqsubseteq B$ and add it to the closure. Since the axiom was not already in the closure, we check whether we can apply any rules. The only rule that applies is the $R_{\sqsubseteq}$ rule. Given the premise $A \sqsubseteq A$ and the side condition $A \sqsubseteq B$, we can trivially derive that $A \sqsubseteq B$, which is already in the closure, and move to the next axiom. As before, we will report the rest of the classification in list form.

1. Next axiom in Todo: $B \sqsubseteq C$
   (a) Insert axiom into closure.
   (b) Since it was new (i.e. not already present in the closure), apply all inference rules.
   (c) The first rule that applies is the $R_{\sqsubseteq}$ rule with the premise $B \sqsubseteq B$. Since the consequence is already in the closure, we move to the next rule.
   (d) The second rule that applies is again the $R_{\sqsubseteq}$ rule, this time with the premise $A \sqsubseteq B$. The consequence $A \sqsubseteq C$ is not in the closure yet, so it is added to the todolist.
2. Next axiom in Todo: $A \sqsubseteq C$
   (a) Insert axiom into closure.
   (b) Since it was new (i.e. not already present in the closure), apply all inference rules.
   (c) The first rule that applies is the $R_{\sqsubseteq}$ rule with the premise $A \sqsubseteq A$. Since the consequence is already in the closure, we move to the next rule.

(d) No further rules apply.

3. Since the todolist is empty, classification is complete.

Enhanced Traversal vs Consequence-based classification

Consequence-based approaches are often associated with better performance because they preclude any need to check possible *non-subsumptions*, while those based on traversal and tableau often involve predominantly *negative* subsumptions tests. Consequence-based approaches are most commonly employed by reasoners that deal only with the fragments of $\mathcal{SROIQ}$ (and therefore OWL 2 DL) such as $\mathcal{EL}$ and $\mathcal{EL}++$, and are used by some popular reasoners such as ELK and CEL. Tableau-based approaches on the other hand are utilised for expressive fragments of OWL 2 DL, i.e. those extending $\mathcal{ALC}$.

The worst case is uninformative with respect to real performance (tractable does not imply practical). If we consider a naive classification algorithm (n^2), a reasonably large ontology such as SNOMED ($n = 300,000$) and assume every test is tractable and takes 0.5 milliseconds, it would still take around 500 days to fully classify SNOMED.[10] This is obviously *impractical*, despite using a tractable algorithm. Another argument, the avoidance of non-subsumptions [Kaz09], is also only of limited use. First of all, tableau reasoners also employ means that avoid a large number of non-subsumptions (*and* subsumptions). Second, the same inference can be drawn multiple times (duplication) during consequence-based classification [KKS14]. And lastly, inferences are drawn that do not relate to atomic subsumptions (redundancy) during consequence-based classification. These factors need to be taken into account when determining which of the two is *generally* better, and requires carefully crafted empirical investigations.

2.3.3 Optimisations

Most of what we call *reasoner development* is about crafting, combining and verifying the viability of *optimisations* to improve reasoning performance. Some optimisations are tailored to particular ontologies [MJL13], some apply only to a very specialised setting,[11] some apply only to particular DL fragments and most only make sense in the context of a particular calculus. There are four types

[10] Example adapted from slides by Pavel Klinov and Bijan Parsia for ESSLLI 2013, `http://esslli2013.de/accepted-courses/`

[11] For example Targeted Communication [MHM13] for Distributed Description Logics

of optimisations that are of special interest to improving classification performance [BCM+07]:

1. *Pre-processing optimisations* try to rewrite the ontology in such a way that classification and subsumption testing becomes easier.

2. *Satisfiability optimisations* try to make tests easer by optimising the satisifiability engine.

3. *Subsumption test optimisations* try to make tests faster by replacing them, either fully or partially, by cheaper ones.

4. *Traversal optimisations* are targeted at avoiding subsumption tests, for example by exploiting the typical tree-shape of the class hierarchy.

Three well known members of the family of *pre-processing optimisations* that are usually employed by traversal and tableau-based approaches are *normalisation*, *simplification* and *absorption*. Normalisation for example rewrites axioms in such a way that the satisfiability engine can detect clashes early on. This is particularly important because we can describe the same thing in different ways, for example $A \sqcap B$ and $B \sqcap A$, as well as $\neg(A \sqcap B)$ and $\neg A \sqcup \neg B$. Absorption is an optimisation that attempts to reduce the high degree of non-determinism induced by general concept inclusions, for example by rewriting axioms of the for $A \sqcap B \sqsubseteq C$ to an axiom with an atomic left hand side: $A \sqsubseteq C \sqcup \neg B$.

One of the most important optimisations that is aimed at *making tests easier by optimising the satisfiability engine* is back-jumping. When exploring potentially deeply nested non-deterministic branches in a tableau setting, the algorithm generally retracts to the last non-deterministic choice and continues with the next branch. It is easily possible that the clash was caused by an interaction that was independent of the expression that caused the non-deterministic choice, which may result in it happening again. Back-jumping aims at recognizing such situations and enables retracting to a non-deterministic choice that would avoid the same clash happening again [BCM+07].

Among the optimisations for *making tests easier by replacing them with cheaper tests* is *caching of partial expansion trees*. This optimisation addresses the issue that many (potentially costly) satisfiability checks are repeated with minor variations (as part of other satisfiability tests). Cached partial expansion trees can be used to prove non-subsumptions without actually performing them. Other

optimisations exploit graph properties to replace certain tests by look-ups to the transitive closure of the known subsumptions [BCM+07].

The main sub-category of optimisations that aim at reducing the traversal space, see Definition 2.1, are traversal algorithms. The most important traversal algorithms are those based on the *Enhanced Traversal* [BHN+94] (ET), and more recently the *Novel Approach* [GHM+12] (NA). Another important technique to avoid unnecessary subsumption tests by exploiting the asserted knowledge are told subsumers [TH06].

Definition 2.1. *Given an ontology $\mathcal{O}$, a reasoner $\mathcal{R}$, the set of all pairs of atomic concepts $\mathcal{T}^{\mathcal{O}} = \widetilde{\mathcal{O}}_C \cup \{\top, \bot\} \times \widetilde{\mathcal{O}}_C \cup \{\top, \bot\}$ in $\mathcal{O}$, the sequence of subsumption tests $(ST(A_j, B_j, \mathcal{O}, \mathcal{R}, i))_j$ with $(A_j, B_j) \in \mathcal{T}^{\mathcal{O}}$ conducted by $\mathcal{R}$ during the ith classification run of $\mathcal{O}$ is called the* traversal space *of $\mathcal{R}$ on $\mathcal{O}$ and denoted $\mathcal{T}^{\mathcal{O}}_{\mathcal{R}} \subseteq \mathcal{T}^{\mathcal{O}}$.*

2.4 Modularity

In the following, we will briefly introduce the core notions of modularity we need in order to understand the intuitions behind most of the approaches and experiments presented in this thesis. Some basic concepts and their notations can be found in Table 2.2. The goal of this section is not to provide a full fledged introduction into modularity, but to *motivate our choice* of focusing on (syntactic) locality-based $\bot$-modules. For a more comprehensive introduction, the interested reader is referred to the PhD thesis of Del Vescovo [DV13] and Grau et al. [GHKS08]. We refer to the the ontology from which $\mathcal{M}$ was extracted as the parent ontology, defined as follows:

Definition 2.2. *Given a module $\mathcal{M}$ of $\mathcal{O}$ with $\mathcal{M} \subseteq \mathcal{O}$, we call $\mathcal{O}$ the* parent ontology *of $\mathcal{M}$.*

Conservative extensions are a core notion to understand modularity in logics. We say that two interpretations $\mathcal{I}$ and $\mathcal{J}$ coincide on a signature Σ (notation: $\mathcal{I}|_\Sigma = \mathcal{J}|_\Sigma$) if $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$ and $X^{\mathcal{I}} = X^{\mathcal{J}}$ for every $X \in \Sigma$ and call $\mathcal{I}|_\Sigma$ the restriction of the interpretation $\mathcal{I}$ to the terms in Σ.

Definition 2.3 (Grau *et al.*, [GHKS08])**.** *Let $\mathcal{O}$ be a $\mathcal{SROIQ}$-ontology, $\mathcal{M} \subseteq \mathcal{O}$, and Σ a signature. We say that:*

Table 2.2: Important terminology related to modularity, used throughout the thesis.

Term	Denoted by	Definition
Module	$\mathcal{M}$	Subset of an ontology which provides a number of logical guarantees over its signature.
Seed signature	Σ	A set of OWL entities.
Module extraction	x-mod$(\Sigma, \mathcal{O})$	Extraction of an x-module (such as $\top$ or $\bot$, as explained later on) with respect to a seed signature Σ and an ontology $\mathcal{O}$

1. $\mathcal{O}$ *is a* deductive Σ-conservative extension (Σ-dCE) *of* $\mathcal{M}$ *if, for all* $\mathcal{SROIQ}$*-axioms* α *with* $\widetilde{\alpha} \subseteq \Sigma$*, it holds that* $\mathcal{M} \models \alpha$ *if and only if* $\mathcal{O} \models \alpha$*;*

2. $\mathcal{O}$ *is a* model Σ-conservative extension (Σ-mCE) *of* $\mathcal{M}$ *if* $\{\mathcal{I}|_\Sigma \mid \mathcal{I} \models \mathcal{M}\} = \{\mathcal{J}|_\Sigma \mid \mathcal{J} \models \mathcal{O}\}$*;*

3. $\mathcal{M}$ *is a* dCE-based (mCE-based) module for Σ *of* $\mathcal{O}$ *if* $\mathcal{O}$ *is a* Σ*-dCE (*Σ*-mCE) of* $\mathcal{M}$*;*

4. If $\mathcal{M}$ *is a dCE-based module for* Σ*, we also say that* $\mathcal{M}$ covers *or* provides coverage to $\mathcal{O}$ *for* Σ.

(1) Model-conservativity means that all models of a module can be extended to a model of their parent ontology. (2) Deductive conservativity means that anything entailed by a module with respect to a signature is entailed by the whole ontology. (3) Model conservativity implies deductive conservativity, but not the other way around.

Deciding whether $\mathcal{M} \subseteq \mathcal{O}$ is a conservativity-based module of $\mathcal{O}$ is undecidable even for simple logics [LWW07]. Locality-based modules were suggested as computationally feasible alternatives to minimal deductive conservative extension-based modules: they are over-approximations of the latter, i.e. they may contain superfluous axioms not needed to preserve entailments over a signature.

There are numerous kinds of locality-based modules, all providing coverage, but with different properties [SSZ09]. It has been shown that, despite being approximations, *syntactic* locality-based modules are rarely different from semantic locality-based modules in practice [VKP+13]. Syntactic locality-based modules come in three flavours: $\bot$, $\top$ and $\star$. All come with some unique properties [DV13]:

- $\bot$-modules guarantee that if $\mathcal{O} \models A \sqsubseteq B$ and $A \in \widetilde{\mathcal{M}}$, that $B \in \widetilde{\mathcal{M}}$.
- $\top$-modules guarantee that if $\mathcal{O} \models A \sqsubseteq B$ and $B \in \widetilde{\mathcal{M}}$, that $A \in \widetilde{\mathcal{M}}$.

- $\star$-modules are very small, as they are the consequence of repeatedly extracting a $\bot$-module from a $\top$-module and so on.

In this thesis we are concerned with syntactic $\bot$-modules because (1) they are usually smaller than $\top$-modules and (2) the set of all $\star$-modules in $\mathcal{O}$ does not have the property of *classification completeness*, see Definition 2.4. $\top$-modules are typically larger because for every $A \in \widetilde{\mathcal{M}}_C$ it is guaranteed that $B \in \widetilde{\mathcal{M}}_C$ if $\mathcal{O} \models A \sqsubseteq B$. In other words, for every name in the signature, we are guaranteed to find also all its subclasses in the signature. The converse is true for $\bot$-modules. Since a class has typically more subclasses than superclasses, $\top$-modules tend to be larger. As we will see later, the set of genuine $\star$-modules in $\mathcal{O}$ lack our completeness criterion with respect to classification (Definition 2.4), and are therefore not suitable for use in classification.

Logical modules in general and locality-based $\bot$-modules in particular come with a range of properties. The following properties are relevant for this thesis:

- *covering* if $\forall A, B$ with $\widetilde{A} \cup \widetilde{B} \subseteq \Sigma$: $\mathcal{O} \models A \sqsubseteq B \leftrightarrow \mathcal{M} \models A \sqsubseteq B$
- *self-contained* if $\forall A, B$ with $\widetilde{A} \cup \widetilde{B} \subseteq \Sigma \cup \widetilde{\mathcal{M}}$: $\mathcal{O} \models A \sqsubseteq B \leftrightarrow \mathcal{M} \models A \sqsubseteq B$
- *depleting* if $\forall A, B$ with $\widetilde{A} \cup \widetilde{B} \subseteq \Sigma \cup \widetilde{\mathcal{M}}$ with $\emptyset \not\models A \sqsubseteq B \rightarrow \mathcal{O} \setminus \mathcal{M} \not\models A \sqsubseteq B$
- *subsumer-complete* if $\forall A \in \widetilde{\mathcal{M}}$ and $\forall B \in \widetilde{\mathcal{O}}$: $\mathcal{O} \models A \sqsubseteq B \rightarrow \mathcal{M} \models A \sqsubseteq B$.

It follows that if $\mathcal{M}$ is a self-contained Σ-module, then it covers Σ. Coverage can be phrased as capturing all the ontology's knowledge about the *seed signature*. Depletingness can be informally phrased as: $\mathcal{O} \setminus \mathcal{M}$ knows nothing about the signature of $\mathcal{M}$.

$\bot$-modules in particular come with a very important property that we informally call *subsumer-completeness*. Subsumer-completess guarantees that, for every concept name $A \in \widetilde{\mathcal{M}}$, B is also in the signature of the module if $\mathcal{O} \models A \sqsubseteq B$. The most important limitation of locality-based modules is that they are not minimal, that is, they may contain superfluous axioms that to not contribute to preserving entailments over its signature. This means that in practice, locality-based modules may be larger than strictly necessary.

2.4.1 Types of Modules and Module Extraction

There are various kinds of computationally feasible approaches to modularity for OWL or its underlying DLs. For a comprehensive survey of module types the interested reader is referred to Chiara Del Vescovos PhD Thesis [DV13].

One motivation to employ $\bot$-modules is their relatively small size. A recently

developed approach based on rules allows the extraction of modules that are on average much smaller than locality-based ones [RKGH15]. The authors point out that the use of $\bot$-modules is stricter than necessary in order to ensure that given a module $\mathcal{M} \subseteq \mathcal{O}$, for every name $A \in \widetilde{\mathcal{M}}$, if $\mathcal{O} \models A \sqsubseteq B$ then $\mathcal{M} \models \mathcal{O}$. While modules are smaller, which could help decreasing classification time, extracting them is not significantly more efficient. It remains to be seen how this approach will fare in practice.

Another novel approach for module extraction is AMEX [GKW14], a module extraction algorithm for expressive acyclic DL ontologies. While the work done around AMEX modules certainly contributes to the understanding of lower and upper approximation (for example that the lower approximation often coincides with the upper one), and is making some ground towards proving that an approximation algorithm for minimal depleting modules can be *almost* empirically optimal, the extraction techniques are too new to be of much use for practical applications. The approach does result in smaller modules, even compared to $\top\bot$-modules, but it neither appears to be very efficient (extraction time wise), nor is it clear how suitable it is for anything beyond acyclic $\mathcal{ALCQI}$.

2.5 Classification-preserving Decompositions

The concept of *classification-preserving decomposition* is central to this thesis, and simply defined as follows:

Definition 2.4. *A set* $D = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \ldots \cup \mathcal{O}_n$ *is called a* classification-preserving decomposition *of* $\mathcal{O}$ *if each* D_i $\bigcup_i CH(\mathcal{O}_i) = CH(\mathcal{O})$. *Any such* D_i *is called a* chunk *in the decomposition* D.

In other words, a decomposition is a set of subsets of $\mathcal{O}$ that, if classified one by one, would result in a set of class hierarchies that could be merged to equal the class hierarchy derived from $\mathcal{O}$. We call this property *classification completeness*. In the course of this thesis, any use of the term *decomposition* refers to the concept of *classification-preserving decomposition*, unless otherwise indicated. *Modular decompositions* are special cases of classification-preserving decompositions, defined as follows:

Definition 2.5. *A set* $D = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \ldots \cup \mathcal{O}_n$ *is called a* modular decomposition *of* $\mathcal{O}$ *if each* D_i *is a module of* $\mathcal{O}$ *and* $\bigcup_i CH(\mathcal{O}_i) = CH(\mathcal{O})$.

In Chapter 7 we will conduct an experiment involving both modular and non-modular *classification preserving decompositions*. In order to ensure that non-modular decompositions are classification complete, they are *based* on modularity. Therefore, we call them *module-based decompositions*. Module-based decompositions can be seen as *summaries* of modular decompositions. In practice, we might choose to combine sets of (for example heavily intersecting) modules to *chunks*, which may or may not be modules. Such summaries preserve classification completeness because the original modular decomposition (lets say D) preserved it. Adding axioms to any $D_i \in D$ (for example through a merge with other chunks) preserves any atomic subsumption $\eta \models D_i$ because of monotonicity. Since there was at least one module in the original modular decomposition that preserved the entailment, we have to have at least one *chunk* that entails it.

It is very important to distinguish our terminology from other incomparable uses. The most important distinction is between our notion of decomposition and its use around the *Atomic Decomposition*. Many of our decompositions are *based on* the Atomic Decomposition (AD). However, the AD itself is not a decomposition in our sense: It is a set of *atoms* rather than modules, and as such, as we will see in the next section *represents* (a potentially large number of) classification preserving decompositions. The *Distributed Description Logic* community also uses somewhat similar terminology. For example, their use of syntactic modularity refers to the need to organise large ontologies in manageable and compact modules, and offers provisions to control the syntactic interactions between these modules [BVSH09]. The notion of a *signature decomposition* [KLPW10] of $\mathcal{O}$ is also a related, but the unions of the chunks resulting from this decomposition are not necessarily a subset of the whole ontology; the union is merely required to be *logically equivalent* to $\mathcal{O}$.

In the following, we will describe two families of modular decompositions. We will omit technical details, most specifically all proofs and formal definitions and focus on key properties, rationale, implementations and use cases.

2.5.1 Decompositions Based on the Atomic Decomposition

The Atomic Decomposition (AD) of an ontology $\mathcal{O}$ can be viewed as a succinct representation of *all modules in an ontology* [DVPSS11a]. The structure of the

AD is represented by a dependency graph whose nodes are atoms. An atom is a maximal disjoint subset of an ontology such that their axioms either appear always together in modules, or none of them does (for a more formal definition see [DV13]). The set of atoms is a *partitioning* of $\mathcal{O}$ such that the union of all atoms correspond to the whole ontology[12] and each atom corresponds to a *module*. Therefore, as mentioned earlier, the Atomic Decomposition is not itself a decomposition according to Definition 2.4. Any module represented by an atom can be simply materialised by computing the union of all axioms in all atoms directly and indirectly dependent on the atom. Unions of atoms, including their dependencies, *do not necessarily correspond to modules*. This is an important fact to keep in mind during the discussion of their potential for modularity-based reasoning in the next chapter.

We distinguish between four (overlapping) types of atoms in this thesis:

1. An atom without outgoing edges is called *maximal* or *root* atom.
2. An atom without incoming edges is called *minimal* or *leaf* atom.
3. An atom with incoming and outgoing edges is called *intermediate* or *branch* atom.
4. An atom with no incoming and no outgoing edges can be referred to as maximal, root, leaf, minimal, *min-max* or *root-leaf*.

We use the same adjectives for describing the corresponding modules. For example, a *maximal module* (or root module) refers to the module corresponding to a *maximal atom* (or root atom).

We can see an example of the Atomic Decomposition of a known toy-ontology in Figure 2.4. According to our atom types, atoms 10, 14, 15, 16, 17 and 18 are *root* atoms and atom 0 is the only *leaf* atom. Note that the Atomic Decomposition *partitions* the ontology, i.e. no axiom is shared between any two atoms. The Atomic Decomposition itself does not preserve the relationship between signatures and the atoms. Because of that, it may be enriched with labels that represent the *Minimal Seed Signature* Σ, i.e. the minimal set of of terms such that the module corresponding to it is exactly represented by the atom and all its dependencies. In other words, given an atom $\mathcal{A}$ and a label Σ for $\mathcal{A}$, the x-mod$(\Sigma, \mathcal{O})$ would result exactly in the genuine module module corresponding to $\mathcal{A}$. Labelled Atomic Decompositions are extensively discussed elsewhere [DV13].

Two kinds of axioms are worthy of further consideration: *syntactic tautologies*

[12]With the exception of *syntactic tautologies*, which do not occur in any atom.

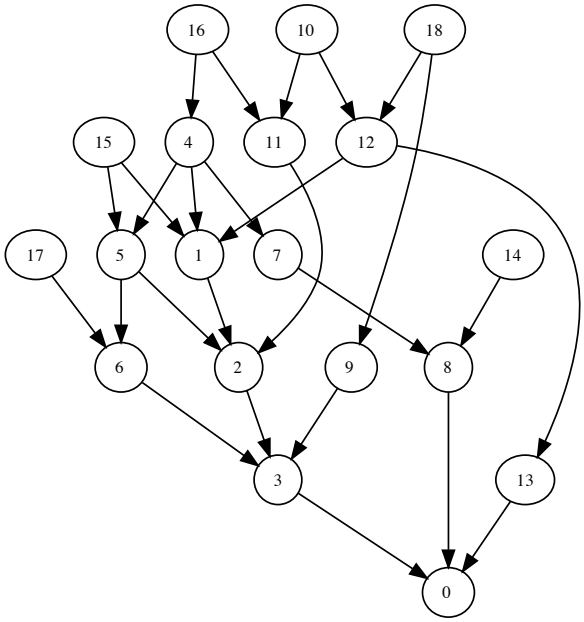


Figure 2.4: An example for the ($\perp$) Atomic Decomposition of the Koala ontology, from [DV13].

and *global axioms*. Syntactic tautologies (such as $A \sqsubseteq \top$) are not part of any module because they are *always local*, regardless of the signature. Therefore, the set of the axioms in an ontology is a *superset* of the module of $\widetilde{\mathcal{O}}$ and consequently all axioms that are mentioned in any given decomposition, which is a fact that is particularly important when counting axioms as part of an experiment. Global axioms are axioms that occur in each module, including the module for the empty seed signature. In the literature, it is (without loss of generality) often assumed that $\mathcal{O}$ contains neither global axioms nor syntactic tautologies [DVPSS11a]. This is not realistic for ontologies on the web: they often contain both. Global axioms can have particularly important consequences for the shape of the Atomic Decomposition, the most important being that they generally have only a single leaf atom. Examples of global axioms are all ABox axioms, and some axioms involving nominals, for example $\top \sqsubseteq \{a\} \sqcup \{b\}$.

The original idea behind the definition of the Atomic Decomposition is to make explicit the modular structure of an ontology without having to somehow represent the set of all its potentially exponentially many modules. Every module that is represented by an atom is a *genuine module*, formally defined elsewhere [DV13]. Genuine modules are in essence modules that cannot be decomposed further into sets of modules. Note that the Atomic Decomposition is defined for all three popular variants of locality-based modules ($\bot, \top, \star$), and the choice of module type has a strong effect on the shape of the decomposition. For example, as $\top$-modules tend to be bigger than $\bot$-modules, we usually have *fewer* modules in decompositions that are derived from $\top$-decompositions. As described earlier, decompositions based on $\star$-modules are not necessarily *classification-complete*. Therefore, we focus in this thesis on decompositions based on $\bot$-modules.

The Atomic Decomposition was originally proposed as a tool for ontology engineers to "understand its topicality, connectedness, structure, superfluous parts, or agreement between actual and intended modeling" [DVPSS11a]. Suggested use cases, discussed extensively in the PhD thesis of Del Vescovo [VKP+13], were module count, offline module extraction (as a form of pre-computing all modules), evaluation of modelling pattern extraction methods and ontology comprehension. It also quickly became apparent that the modular structure exposed by the Atomic Decomposition could be used to optimise reasoning, by providing a tractable means to produce a set of classification-complete modules that in its entirety entail all the logical consequences of $\mathcal{O}$ [TP12a, VKP+13] - the sort of

rationale that is also underlying our own work.

Several algorithms to extract the Atomic Decomposition have been suggested [DV13, MRW14, Tsa12]. Despite the tractability of the underlying decision procedure, current implementations are in a prototypical state, and tend to be very slow especially with respect to large ontologies - the ones that modular reasoning promises to help out with in particular. However, work is being done to make the extraction more efficient. The work by Martin-Recuerda et al. is currently under Major Revision (Semantic Web Journal) and promises a significant speed-ups ("staggering speedup of over 1,000 times") at least for OWL 2 EL ontologies [MRW15]. The original proposal of the authors algorithms can be found in their ISWC publication [MRW14], but there are some concerns with respect to the completeness of the implementation.[13] Details of the implementation and the improved variant of the original decomposition algorithm were described in a work by Tsarkov [Tsa12]. In this thesis, we use the OWL API tools implementation, which is the slowest of the existing ones. However, the FaCT++ implementation [Tsa12] for example does not deal with ontologies involving datatypes, and the Martin-Recuerda approach was not available at the time, and was also never evaluated independently from the authors.

2.5.2 The MORe Decomposition

The MORe decomposition[14] consists of exactly two modules: the $\mathcal{L}$ module and the *remainder* module [RGH12]. The approach was designed for use in modular reasoning. The rationale behind the MORe decomposition is to compute a large $\mathcal{L}$-module, which can be processed using a hopefully efficient classification algorithm, so that only the remainder module needs to be classified with a full fledged OWL 2 DL reasoner.

The decomposition algorithm takes as an input an ontology $\mathcal{O}$, and attempts to compute a hopefully large signature Σ such that $\bot$-mod$(\Sigma, \mathcal{O})$ is in $\mathcal{L}$. The algorithm initialises Σ with $\widetilde{\mathcal{O}} \setminus \widetilde{\mathcal{O}}_{\overline{\mathcal{L}}}$, i.e. the signature of all the axioms that are in $\mathcal{L}$, and then gradually reduces it until the module resulting from $\bot$-mod$(\Sigma, \mathcal{O})$ is in $\mathcal{L}$. Details of the algorithm can be found in the MORe system description [RGH12].

[13] The reviewers of the aforementioned JWS submission complain that the implementation (not the algorithm) is incorrect w.r.t the dependency edges.

[14] Note that the authors of MORe would probably not refer to it as a decomposition.

The algorithm is not restricted to a particular $\mathcal{L}$. In practice, it was used to extract an ELK-module, i.e. a module in a language for which the ELK classification algorithm is sound, complete and terminating. The ELK reasoner [KKS14] is known to be very efficient for OWL 2 EL ontologies. The remainder module then contains the part of the ontology that involves axioms beyond the ones processable by ELK. More details about the use of the MORe decomposition for classification can be found in the following chapter. We are currently not aware of any alternative implementations.

In contrast to the Atomic Decomposition, the algorithm is *not universally applicable* to all OWL 2 DL ontologies. A successful decomposition would be indicated by being comprised of exactly two non-empty modules. This may not be true in at least two cases: (1) the input ontology is already in $\mathcal{L}$ and (2) the current implementation does not (yet) cover the full extend of OWL 2 DL. For example, the last available implementation of the decomposition algorithm in the MORe reasoner does not deal with nominals and ABoxes, i.e. it will return an empty Σ for ontologies with nominals/individuals.

2.6 Summary

In this chapter, we have introduced some of the core concepts we make use of in our work. In particular:

- We have described in detail the reasoning task of classification and contrasted two important approaches. We have emphasised that while the procedures that are underpinning Description Logic reasoning are sound, complete and terminating, existing implementations may be buggy, often due to the high complexity of reasoner architecture and intertwining optimisations. We also tried to convince the reader that worst case results may not be indicative of the actual performance of OWL reasoners. In Chapter 4, we will introduce a framework that will allow us to observe the performance of OWL classification empirically.
- We have introduced our notion of classification-preserving decompositions (CPD). CPD's are sets of subsets of the ontology, which, if all were to be classified independently, would include the entire set of atomic subsumptions entailed by the ontology. In the next chapter, we will describe how CPD's are used for modular reasoning.

- We have motivated our choice of syntactic locality-based $\bot$-modules for classification-preserving decompositions. Modules based on syntactic locality are cheap to extract. $\bot$-modules in particular are subsumer-preserving and can therefore be used to construct CPD's.

Chapter 3

Reasoning with Locality-based Modules in Description Logics

In this chapter we will discuss the different ways in which modules are used for reasoning, describe existing implementations in detail, and gather arguments for and against using modules and classification-preserving decompositions for optimising Description Logic reasoning. The main goals of this chapter are:

- Introducing the reader to the applications that make use of modules in the context of reasoning.
- Conveying a clear understanding of the *potential benefits* as well as *threats and limitations* of employing modules and module-based decompositions to improve reasoning performance.
- Isolating the important research questions that need to be answered.

3.1 Terminology and Models

In Table 3.1, we introduce some of the terminology relevant to this chapter that we will continue to use throughout this thesis.

3.1.1 Reasoner with Modularity-sensitive Calculus vs Modular Meta-reasoning Framework

Before we introduce our model of modular reasoning, we want to distinguish two views on the architecture of modular reasoning frameworks: The reasoner with a modularity-sensitive calculus and the modular meta-reasoning framework.

A *modular meta-reasoning framework* introduces a new layer on top of the typical OWL reasoner architecture. This layer deals with the decomposition of the ontology into a set of chunks (modules or sets of modules), and implements a

Table 3.1: Important terms used throughout the thesis (continuation of Table 2.1).

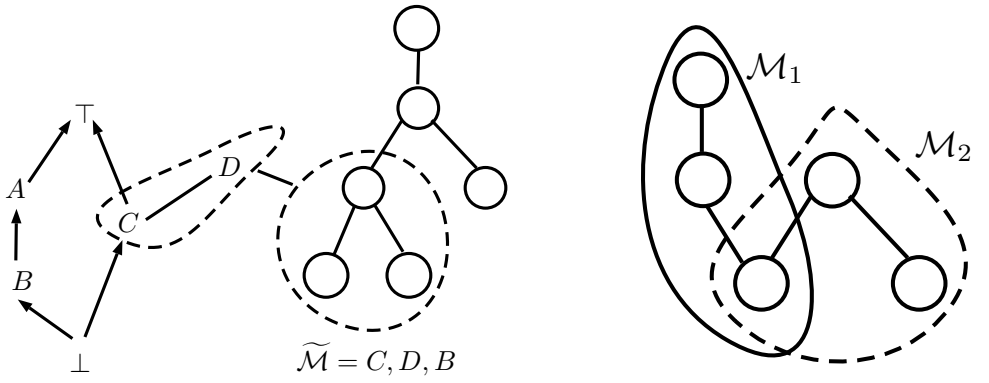
Term	Denoted by	Definition
Module	$\mathcal{M}$	Syntactic locality-based $\bot$-module of ontology $\mathcal{O}$ with $\mathcal{M} \subseteq \mathcal{O}$.
Subsumption test for $A \sqsubseteq B$	$ST(A, B, \mathcal{O})$	A subsumption test between A ad B as triggered as part of a traversal based classification procedure (such as Enhanced Traversal / Tableau) of $\mathcal{O}$.
Subsumption test measurement for $A \sqsubseteq B$	$ST(A, B, \mathcal{O}, \mathcal{R}, i)$	A subsumption test measurement between A ad B by reasoner $\mathcal{R}$ in $\mathcal{O}$, with i denoting the run.
Classification time	$CT(\mathcal{O}, \mathcal{R})$	The time it takes to classify an ontology $\mathcal{O}$ by a reasoner $\mathcal{R}$.
Ontology language	$\mathcal{L}(\mathcal{O})$	The logical language in which an ontology is expressed, for example $\mathcal{EL}$ or OWL 2 EL.
Justification	$\mathcal{J}^{\alpha}_{\mathcal{O}}$	The set of all justifications of an entailment α with respect to ontology $\mathcal{O}$, i.e. a minimal subset of $\mathcal{O}$ that entails α.
Monolithic reasoner	none	A reasoner that performs classification on the whole ontology rather than on its classification-preserving decomposition.
Modular reasoner	none	A reasoner that performs reasoning on a set of modules in the ontology.
Decomposition-based reasoner	none	A reasoner that performs classification on a classification-preserving decomposition of the ontology.

strategy to traverse those chunks one by one. Each chunk is processed, i.e. classified, independently by dispatching it to a delegate reasoner (a standard OWL reasoner) and merging the results. The key here is that the delegate reasoners are treated like blackboxes, i.e. they do not communicate results with each other.

The *reasoner with a modularity-sensitive calculus* is a monolithic reasoner that implements its own classification procedure, i.e. the modular structure of the ontology will be exploited directly to prune the subsumption test (traversal) space and the search space of the satisfiability checking. For example, a modularity-sensitive satisfiability engine will only ever consider those GCI's[1] that are really modularly relevant (currently GCI rules are often applied even when unnecessary). In order to be fully modularity-sensitive, reasoner developers need to integrate the modular structure of the ontology tightly with the rest of the classification process, e.g. on the level of Enhanced Traversal (avoid tests based on module properties, as discussed later in Section 3.4.2) or the satisfiability engine (modularity-sensitive application of GCI rules, etc.). Chainsaw, for example, currently implements a two level traversal: The outer loop iterates over the atoms of the Atomic Decomposition, and for each module corresponding to an atom, the Enhanced Traversal is triggered to classify all concepts in the label of that atom (explained in more detail in Section 3.3.2). Subfigure 3.1a shows a possible interaction between Enhanced Traversal and Atomic Decomposition with respect to a potential subsumption test between C and D. This interaction can have many forms: for example, it could be checked whether any module signature contains both class names. If this is not the case, the test can be avoided. If a subsumption test cannot be avoided, modularity could ensure that it is triggered within the bounds of a (ideally as small as possible) module, thereby, as we will see later in Section 3.4.1, reducing the amount of irrelevant axioms to consider and potentially making the test faster.

In this thesis, we are primarily concerned with *modular meta-reasoning frameworks*. Subfigure 3.1b shows a classification-preserving decomposition of an ontology (based on the Atomic Decomposition) into two modules. The modular meta-reasoning framework would make a decision to which of a number of delegate reasoners $\mathcal{M}_1$ would be dispatched, and fully classify it. Then it would move on to $\mathcal{M}_2$. As a final step, it would query the classified modules to construct the

[1] A GCI or General Concept Inclusion is an axiom of the form $C \sqsubseteq D$ were C and D are (potentially complex) concepts.



Enhanced Traversal — Modular Structure of $\mathcal{O}$

(a) Reasoner with a modularity-sensitive calculus

(b) Modular meta-reasoning framework

Figure 3.1: Types of modular reasoners. The modular structure of $\mathcal{O}$ is in both cases represented as its Atomic Decomposition, i.e. the circles represent atoms and the edges top-down dependencies.

full class hierarchy. This approach does not require any *commmunication between delegate reasoners*. An important consequence of the absence of communication is the possibility of redundancy caused by overlapping modules. For example, consider again Subfigure 3.1b: Reasoning with respect to the atom shared between $\mathcal{M}_1$ and $\mathcal{M}_2$ would have to be done twice (both when classifying $\mathcal{M}_1$ and $\mathcal{M}_2$). If this atom would correspond to 95% of the parent ontology of $\mathcal{M}_1$ and $\mathcal{M}_2$, this redundancy could cause an (almost) two-fold increase in classification time.

However, it does make sense to contemplate modular meta-reasoning frameworks that are at least to some extent glassbox: If a delegate reasoner processing a module would be aware of what work has been done before, much of the negative effect caused by the redundancy could be mitigated. In a setting involving Enhaced Traversal (or similar) the vast majority of subsumption tests turn out to be negative. This raises the open question of how non-subsumptions should be represented in order to be effectively communicated. A HermiT-like *shared* possible subsumption relation might be a possibility [GHM+14], requiring however an extension to the current monolithic reasoner implementations. In Chapter 7 of this thesis, we take the pure blackbox model as a baseline and simulate alternative views, such as a simple glassbox model.

3.2 Model of Modular Classification

While some of our results in this thesis are applicable to modularity-sensitive calculi, our attention lies on *modular meta-reasoning frameworks.* In the following, we will describe our model of modular classification as part of such a meta-reasoning framework. A modular reasoner in this sense has three core components: a classification-preserving decomposition, a set of delegate reasoners to pick from to process chunks of that decomposition, and a strategy to determine which chunks are assigned to which reasoner and in which order they should be processed. The core components of our model of modular classification are listed in Table 3.2.

Table 3.2: Core concepts around modular reasoning.

Term	Denoted by	Definition
Classification-preserving decomposition	$D_{\mathcal{O}}$	See Section 2.5.
Delegate Reasoner	$\mathcal{R}$	The reasoner that is employed by the modular reasoning approach to perform reasoning on a particular chunk of $D_{\mathcal{O}}$.
Delegation (Process)	$Del(\mathcal{O}, \mathcal{R})$	The process of assigning a chunk to delegate reasoner.
Decomposition (Process)	$Dec(\mathcal{O})$	The process of decomposition defines how to compute a particular (classification-preserving) decomposition from an ontology $\mathcal{O}$.
Classification Strategy	none	Procedure specifying: • Division of decomposition into processable chunks. • Rules for delegation decisions. • Order of the classification of the chunks.

The process of decomposition-based classification is described in the following, presented as a simple enumeration rather than pseudocode because implementations can differ significantly:

Procedure 3.1. *Decomposition-based classification of an ontology $\mathcal{O}$*

1. *Preprocessing involving:*

- Modular *Decomposition: Decomposing $\mathcal{O}$ into a set of modules*
- *Initial Consistency Check: Making sure that $\mathcal{O}$ is consistent, either by checking the consistency of $\mathcal{O}$ directly or the consistency of one of its modules.*
- *Chunking (optional): Summarizing a given modular decomposition into another classification-preserving decomposition.*

2. *Ordering: Deciding an order in which the chunks are processed.*
3. *Delegation: Dispatching chunks in the decomposition to delegate reasoners.*
4. *Classification: Classifying the chunks iteratively as specified by their order (and delegation).*
5. *Result Merging: Producing a unified class hierarchy.*

Figure 3.2 exemplifies *the key aspects* of the process. The reasoner first decomposes the ontology $\mathcal{O}$ into a set of potentially overlapping and potentially interrelated modules (DEC), before it conducts the initial consistency check (CC). Then it (optionally) summarises the modular decomposition into chunks, either resulting again in a modular decomposition, or in a modularity-based decomposition. From the set of chunks, the reasoner computes an ordered list of delegations, i.e. a pair of a chunk and a particular delegate reasoner that the chunk should be dispatched to. In the modular classification stage (MCL), the reasoner iterates through the (ordered) chunks and dispatches them to their assigned delegates (DP). After the delegate classified the chunk (in the fully monolithic fashion, stage labelled MC), the results are added to the class graph (SYNC). During post-processing (POP), the class graph is completed. The delegate reasoner goes through all the usual stages *of a monolithic reasoner*, which will be described in more detail in Section 4.1.

3.3 Applications of Modular Reasoning for OWL Ontologies

While both modularity and reasoning by themselves have been studied extensively, exploiting modules for optimising reasoning in general and classification in particular is a relatively recent development. To date, only a few full fledged OWL reasoners exploit modules for optimising any part of the classification process [MLH+15]. Two current examples are MORe [RGH12] and the experimental

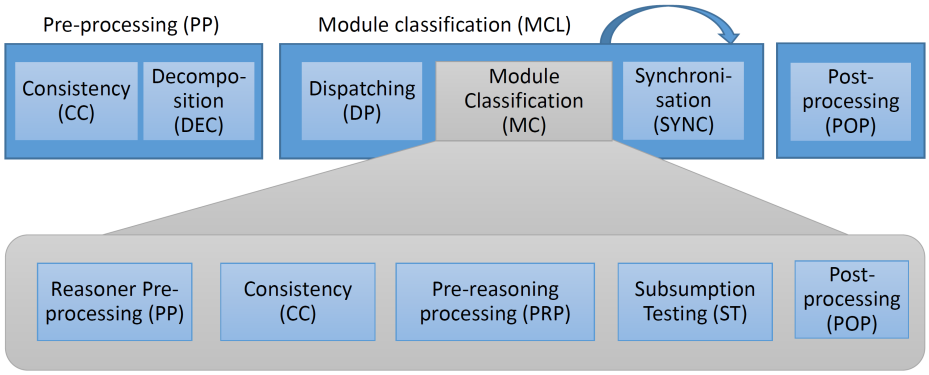


Figure 3.2: Black-box model of decomposition-based classification. Every delegate reasoner in the module classification stage goes through all five stages of the monolithic reasoning model when classifying its assigned module; we are assuming a traversal/tableau style delegate reasoner (consequence-based reasoners follow another stage model).

Chainsaw [TP12a], a prototype for exploiting the Atomic Decomposition for various optimisations for classification. Although both reasoners are designed as *meta-reasoning frameworks*, current implementations are tied to particular delegate reasoners (FaCT++ in the case of Chainsaw and HermiT, Pellet, JFact and ElK in the case of MORe). Another reasoning approach based on identifying *hot spots*, sets of axioms that make classification particularly hard, uses modularity to classify concepts in the signature of the hot spot separately from the rest of the ontology [GPS12]. After we discuss the use of modules to manage the process of *incremental reasoning*, we will briefly discuss a novel approach to *optimise consistency checking* using locality-based modules.

3.3.1 MORe

Big/hard ontologies often have large subsets expressed in an "easy" language, say $\mathcal{L}$, for which we have an efficient $\mathcal{L}$-reasoner (e.g. ELK for OWL 2 EL). The idea of MORe is to compute a (hopefully large) signature $\Sigma^{\mathcal{L}} \subseteq \widetilde{\mathcal{O}}$ that can be used to extract a module $\mathcal{M}$ from the ontology $\mathcal{O}$ that can be fully classified by an efficient $\mathcal{L}$-reasoner, and only classify the, hopefully small, remainder module with a full fledged OWL 2 reasoner, see Section 2.5.2 for a discussion of the MORe decomposition and Section 2.3.2 for a discussion on efficient classification algorithms for the OWL 2 profiles. We can refer back to Subfigure 3.1b for illustration. In the MORe setting, $\mathcal{M}_1$ could be the $\mathcal{L}$-module, to be processed

by ELK, and $\mathcal{M}_2$ the (possibly overlapping) remainder module, processed by HermiT. The core problem addressed by MORe is to make sure that $\Sigma^{\mathcal{L}}$ is as large as possible, so that ideally the $\mathcal{L}$-reasoner does most of the hard work. The algorithm works as follows:

Algorithm 3.3.1: MORe classifyOntology($\mathcal{O}$)

Data: OWL 2 Ontology $\mathcal{O}$
Result: ClassHierarchy $\mathcal{H}$
$\Sigma^{\mathcal{L}} := \mathcal{L}$-signature($\mathcal{O}$);
$\mathcal{M}_{\mathcal{L}} := \bot$-mod($\Sigma^{\mathcal{L}}, \mathcal{O}$);
$\mathcal{M}_{\overline{\mathcal{L}}} := \bot$-mod($\widetilde{\mathcal{O}} \setminus \Sigma^{\mathcal{L}}, \mathcal{O}$);
$\mathcal{H}_{\overline{\mathcal{L}}} :=$ classification($\mathcal{M}_{\overline{\mathcal{L}}}$);
$\mathcal{H} := \mathcal{L}$-classification($\mathcal{M}_{\mathcal{L}} \cup \mathcal{H}_{\overline{\mathcal{L}}}$);

First, the $\mathcal{L}$-Signature is computed. In a nutshell, this is done by gradually shrinking $\widetilde{\mathcal{O}}$ to a signature Σ such that $\mathcal{M}_{\mathcal{L}} := \bot$-mod($\Sigma, \mathcal{O}$) is in $\mathcal{L}$. After extracting the remainder module $\mathcal{M}_{\overline{\mathcal{L}}}$ from the *complement* of the $\mathcal{L}$-signature $\Sigma^{\overline{\mathcal{L}}}$ ($\widetilde{\mathcal{O}} \setminus \Sigma^{\mathcal{L}}$), it is classified by a regular OWL reasoner. In order to save the $\mathcal{L}$-reasoner some work, the hierarchy as determined by the classification of the remainder module $\mathcal{H}_{\overline{\mathcal{L}}}$ is merged with the $\mathcal{L}$-module $\mathcal{M}_{\mathcal{L}}$. Therefore, inferred atomic subsumptions over $\Sigma^{\mathcal{L}} \cap \Sigma^{\overline{\mathcal{L}}}$ do not have to be derived again by the $\mathcal{L}$-reasoner. This set of axioms is classified, and the resulting class hierarchy returned.

In principle, MORe is a modular meta-reasoning framework that allows for arbitrary delegates for the $\mathcal{L}$- and remainder modules. The latest implemented version of MORe[2] (0.1.6 at the time of writing) however is tied to particular implementations. $\mathcal{L}$ is fixed to be the fragment of OWL 2 EL that is processable by the ELK reasoner [KKS14]. The user can choose between JFact, Pellet and HermiT as delegate reasoners for the remainder module. However, as long as there exists a decision procedure for determining whether an axiom is in $\mathcal{L}$, MORe should be easily extendible to allow fully configurable delegate reasoners and MORe-decompositions for any (supported) $\mathcal{L}$. Another caveat of the MORe algorithm is the effect of some kinds of TBox axioms with nominals that cause empty $\Sigma^{\mathcal{L}}$. Axioms of the type $\forall R.A \sqsubseteq \{a\}$ and $\forall R.A \sqsubseteq \exists S.\{a\}$ (and similar) are never $\bot$-local (with respect to any seed signature Σ). As such axioms might not be in $\mathcal{L}$ (but would be part of $\mathcal{M}_{\mathcal{L}}$), the OWL reasoner (despite some axioms

[2] `https://code.google.com/p/more-reasoner/downloads/list`

with nominals being supported by ELK) has to perform the classification of the whole $\mathcal{O}$.

3.3.2 Chainsaw

The idea of Chainsaw is that, for every inference query (subsumption test, instance query), a module $\mathcal{M}$ is created that is small but provides all "information" that is required to answer the query correctly (no loss of relevant entailments) [TP12a, TP12b]. Chainsaw exploits the (Labelled) Atomic Decomposition which it views as a compact representation of all the (locality-based) modules of $\mathcal{O}$, with atoms labelled with the *seed signature* as determined by the labelling procedure. The classification algorithms traverses the atoms in the Atomic Decomposition in a depth first search fashion, and classifies class names in their labels one-by-one. In detail, the Chainsaw classification algorithm works as follows (Algorithm 3.3.2):

Algorithm 3.3.2: Chainsaw classifyOntology($\mathcal{O}$)

Data: OWL 2 Ontology $\mathcal{O}$
Result: ClassHierarchy $\mathcal{H}$
$\mathcal{H} := \{\}$;
$AD :=$ $\bot$-atomic-decomposition($\mathcal{O}$);
$AD :=$ labelAD(AD);
$RA :=$ root-atoms(AD);
for *Atom* **in** RA **do**
    Reasoner = createDelegateReasoner(ad-module(Atom));
    $\mathcal{H}_i :=$ traversal(Atom,Reasoner);
    $\mathcal{H} := \mathcal{H} \cup \mathcal{H}_i$;
end

The algorithm starts with an empty class hierarchy $\mathcal{H}$. First, the $\bot$-Atomic Decomposition AD is computed. Labelling works as follows: we initialise each atom label with the signature of its corresponding module. Then we traverse the AD bottom up, breadth-first, removing from every atom label all entities that are mentioned in any atom dependent on it. Next, the set of root atoms (see Section 2.5.1, RA) is extracted and traversed one by one. Using depth-first search on the AD dependency graph, the algorithms iterates through the class names in the label of each atom, determining their direct superclasses by classifying them in the module corresponding to the atom, and merges the results with $\mathcal{H}$. When

Algorithm 3.3.3: Chainsaw: traversal(Atom,Reasoner)

Data: Atom,Reasoner
Result: ClassHierarchy $\mathcal{H}$
$\mathcal{H} := \{\}$;
for *DependentAtom* **in** *Atom.getDirectDependents()* **do**
  $\mathcal{H}_i :=$ traversal(DependentAtom,Reasoner);
  $\mathcal{H} := \mathcal{H} \cup \mathcal{H}_i$;
end
$\mathcal{H}_i :=$ classifyAtom(Atom,Reasoner);
$\mathcal{H} := \mathcal{H} \cup \mathcal{H}_i$;

Algorithm 3.3.4: Chainsaw: classifyAtom(Reasoner,Atom)

Data: Atom,Reasoner
Result: ClassHierarchy $\mathcal{H}$
$\mathcal{H} := \{\}$;
for *ClassName* **in** *Atom.Label()* **do**
  $\mathcal{H}_i :=$ Reasoner.getDirectSuperClasses(ClassName);
  $\mathcal{H} := \mathcal{H} \cup \mathcal{H}_i$;
end

all atoms are processed, the classification is complete.

While Chainsaw is designed to be a meta-reasoning framework, it seems that a tighter integration with a traversal algorithm could have a significantly positive effect on performance. The current implementation is likely to suffer from a *large amount of redundancy*, for example due to the fact that (1) many of the standard optimisations such as pseudo-model merging or partial expansion trees are currently not shared across delegate reasoners, (2) independent atoms can have intersecting labels, i.e. a class name can occur in several independent atoms, which can cause redundant subsumption testing and (3), perhaps most importantly, current implementations of OWL reasoners suitable for being delegates for Chainsaw perform *full classification* when querying for a particular subsumption. Therefore, the traversal through the labels is *essentially ineffectual* in terms of avoidance. As an example, consider the decomposition depicted in Subfigure 3.1b. Because both $\mathcal{M}_1$ and $\mathcal{M}_2$ share an atom, they also share signature ($\widetilde{\mathcal{M}_1} \cap \widetilde{\mathcal{M}_2}$ is non-empty) and because both are fully classified, subsumption tests are duplicated between all class names $A, B \in \{\widetilde{\mathcal{M}_1} \cap \widetilde{\mathcal{M}_2}\}$.

Another shortcoming of the current implementation is that delegate reasoners

are created *only for root-atoms*. This reduces the potential of the underlying modularity because the search space for any given subsumption is no longer *minimal*, i.e. there are a potentially significantly larger number of subsumptions to check (traversal space) and axioms to scan (search space during tableau construction). The reason for that lies again in the fact that current delegate reasoners *perform full classification*. Given two atoms $a1$ and $a2$ with $a1$ being dependent on $a2$, we would simply redo all the $a2$ work when classifying $a1$. Classifying $a1$ and $a2$ separately therefore makes no sense (as long as the delegate reasoners perform full classification).

The choice of traversing the Atomic Decomposition by depth-first search is just one of a number of possible ways to exploit it (the AD) for modular reasoning. For example, we may choose to classify the connected components one-by-one, see Section 7.2.3 instead. Another straight forward extension to the Chainsaw traversal suggested by the authors (email communication) is to further avoid tests by exploiting the property of *entity-label uniqueness*: if a concept name A occurs in more than one label of a root module, there can be no $B \in \widetilde{\mathcal{O}}$ for which $\mathcal{O} \models A \sqsubseteq B$, as follows from Proposition 1 in Tsarkov et. al [TP12a]. Specialised traversal algorithms such as this one are not part of the investigation presented in this thesis.

The current implementation is also not truly configurable with arbitrary delegate reasoners: There are integrations with FaCT++ and JFact available, but both reasoners are hard-wired into the code. Extending the current version (1.0 at time of writing) of Chainsaw to support configurable delegate reasoners, however, is fairly straight forward. As long as the delegate reasoner implements a routine to obtain direct superclasses of a concept, it can, at least in principle, be used as part of Chainsaw. Note that (again), reasoners such as HermiT, FaCT++, JFact and Pellet *fully classify* the ontology, i.e. all named concepts in the ontology, instead of merely attempting to query for the direct superclasses of a concept at hand when the `getDirectSuperclasses(`A`)` method is triggered. This is often more efficient, especially if the reasoner has to answer more than one such query. In order to mitigate the redundancy, the Atomic Decomposition-based traversal needs to be integrated more tightly with the delegate reasoners internal traversal and optimisations.

3.3.3 Hotspot Reasoning

Another branch of research exploits modules to extract so called *Hot Spots* from ontologies [GPS12]. Hot Spots are fragments of ontologies (typically between 0.03% and 9% of the original size) whose removal have a significant effect on the reasoning time (80% to 99% boost in classification time $CT(\mathcal{O}, \mathcal{R})$ once removed). The intuition behind the Hot Spot extraction is straight forward: We measure the execution time of the individual SAT-tests for all concepts $A \in \widetilde{\mathcal{O}}$, order them in descending order and take the hardest 1000 concepts names as candidates. We then, candidate by candidate, extract their usage, i.e. axioms in which the candidate concept occurs, and extract a $\star$-module $\mathcal{M}$ from $\mathcal{O}$ for the set of entities with which the candidate co-occurs, i.e. the signature of the usage. To verify $\mathcal{M}$ as a Hot Spot, it is made sure that $CT(\mathcal{O} \setminus \mathcal{M}, \mathcal{R}) \ll CT(\mathcal{O}, \mathcal{R})$ (significantly less), see Algorithm 3.3.5 [GPS12].

Algorithm 3.3.5: Hot spots: extractHotSpots(Ontology)

Data: Ontology $\mathcal{O}$
Result: Set S of hot spots in $\mathcal{O}$
$S := \{\}$;
$Candidates := \{\}$;
$Times := \{\}$;
$MAX := 1000$;
for C **in** $\widetilde{\mathcal{O}}_C$ **do**
  $Times := Times \cup < C, SATtime(C) >$;
end
$Times :=$ sortByTimeDescending($Times$);
$Candidates :=$ sublist(0,MAX,$Times$).ClassNames;
for C **in** $Candidates$ **do**
  $U :=$ usage(C);
  $\mathcal{M} := \star$-mod($\widetilde{U}, \mathcal{O}$);
  if $CT(\mathcal{O} \setminus \mathcal{M}, \mathcal{R}) \ll CT(\mathcal{O}, \mathcal{R})$ **then**
    $S := S \cup \{\mathcal{M}\}$;
  end
end

While this novel approach is a useful tool for ontology engineers to enable performance tuning, in its current state it is only suitable for approximate reasoning—the $\star$-module $\mathcal{M}$ does not give any guarantees with respect to finding all subsumers and subsumees,[3] and there are no guarantees at all for $\mathcal{O} \setminus \mathcal{M}$ thus classification will in many cases be incomplete. There are however interesting speed-ups in reasoning time possible. Furthermore this type of investigation can be used to increase our understanding of reasoning performance as it allows us to study relatively small subsets that make reasoning harder.

3.3.4 Module-based Incremental Reasoning

A very typical situation in the process of ontology engineering is that the engineer adds, removes or changes an axiom, and wants to re-classify the ontology to inspect the effect of the changes, for example whether an inconsistency was introduced or in what way the inferred class hierarchy changed as a consequence of a modelling choice. For complex ontologies, in the sense that they are hard to classify, re-classification can can be a bottleneck in the ontology engineering process. The general intuition behind incremental reasoning is that applying small changes to the ontology usually does not require a complete re-classification. In fact, the subsumption relations affected by such a change are often quite small. In order to exploit this intuition, an approach is needed to keep track of which subsumptions *could* be affected by the change. Given an ontology $\mathcal{O}$, a set of additions AX^+, a set of removals AX^- and $\mathcal{M}_A$ being the $\bot$-locality-based module for the seed signature $\Sigma = \{A\}$ in $\mathcal{O}$, the algorithm goes through the following phases, as described in the work by Grau et al. [GHWK07]:

Procedure 3.2. *Incremental Reasoning*

1. *Apply changes: Generate the modified ontology* $\mathcal{O}_m := (\mathcal{O} \setminus AX^-) \cup AX^+$
2. *Process new symbols: Assign to all class names in* AX^+ not *in* $\mathcal{O}$ *a "dummy" module for the empty signature (pretending we do not know anything about the class yet).*
3. *Identify affected modules: We loop through all class names* $A \in \widetilde{\mathcal{O}_m}$ *(including* $\top$*). For any* $\alpha \in AX^-$*, if* α *is non-local with respect to* $\widetilde{\mathcal{M}_A}$*, A is flagged as* negatively affected. *For any* $\alpha \in AX^+$*, if* α *is non-local with respect to* $\widetilde{\mathcal{M}_A}$*, A is flagged as* positively affected.

[3] Given a $\star$-module $\mathcal{M}$ and its parent ontology $\mathcal{O}$, classify$(\mathcal{O} \setminus \mathcal{M}) \cup$classify$(\mathcal{M}) \neq$classify$(\mathcal{O})$.

4. *For each class name in $A \in \mathcal{O}_m$ (including $\top$):*
   (a) *Computing new modules: If A has been flagged as affected in the previous step, its module $\mathcal{M}_A$ is extracted again.*
   (b) *Computing new subsumptions: For each class name in $B \in \mathcal{O}_m$ (including $\bot$), if A was flagged as negatively affected and $\mathcal{O} \models A \sqsubseteq B$ (original ontology) or if A was flagged as positively affected and $\mathcal{O} \not\models A \sqsubseteq B$, then fire the subsumption test for $\mathcal{M}_A \models A \sqsubseteq B$ (else, maintain the original result from $\mathcal{O}$). The key here is that the subsumption test is triggered with respect to the module, thereby potentially reducing the number of irrelevant axioms to search through significantly.*

The underlying proposition providing the core property for module-based incremental reasoning is as follows:

Proposition 3.3.1 (Cuenca Grau *et al.*, [GHWK07], Proposition 3)**.** *Let $\mathcal{O}_1$, $\mathcal{O}_2$ be ontologies, α an axiom, and $\mathcal{M}_\alpha^{\mathcal{O}_1}$, $\mathcal{M}_\alpha^{\mathcal{O}_2}$ respectively modules for α in $\mathcal{O}_1$ and $\mathcal{O}_2$. Then:*

1. *If $\mathcal{O}_1 \models \alpha$ and $\mathcal{M}_\alpha^{\mathcal{O}_1} \subseteq \mathcal{O}_2$, then $\mathcal{O}_2 \models \alpha$*
2. *If $\mathcal{O}_1 \not\models \alpha$ and $\mathcal{M}_\alpha^{\mathcal{O}_2} \subseteq \mathcal{O}_1$, then $\mathcal{O}_2 \not\models \alpha$*

In other words, if an ontology entails α, and the module for α in that ontology is a subset of another ontology, that other ontology has to entail α as well (similar for non-entailment).

Despite the importance of incremental reasoning approaches for managing complex ontologies, our survey of 33 current, stand-alone OWL reasoners (see Section 4.7.1) has revealed only 3 systems whose developers claim to use modules for incremental reasoning [MLH+15]. However, modules are not the only way to achieve incremental reasoning, and 4 other developers have reported to be actively working on integrating incremental reasoning support into their OWL reasoners. For example, Pellet employs *tableau*-tracing to keep track of axioms used for inferred ABox entailments [SPG+07] and ELK uses an approach that does not require any bookkeeping (such as maintaining a modular structure) [KK13]. FaCT++ was recently equipped with facilities for incremental reasoning [Tsa14] based on an updated version of the approach by Grau et al. [GHWK07].

3.3.5 Optimising Consistency Checking Using Modules

Despite being comparatively slow when classifying OWL ontologies compared to other OWL reasoners, Chainsaw regularly outperforms its competition in terms of consistency checking. In the ORE reasoner competition [PMG+15] it took second (2014) and third (2015) place in that discipline. The rationale behind a module-based consistency check lies in the fact that the explanation for an inconsistency has to be present in *every module* in $\mathcal{O}$—even the module of an empty seed signature. The procedure works as follows: First, we extract the module for the empty signature $\mathcal{M}_\emptyset$. If it is empty, we know the ontology is consistent, and we are done. This works because of the following:

Lemma 3.3.2. $\mathcal{O} \models \top \sqsubseteq \bot \iff \forall \mathcal{M} \subseteq \mathcal{O} : \mathcal{M} \models \top \sqsubseteq \bot$

To prove that this holds, we first show the $\Leftarrow$-direction.

Claim: $\mathcal{O} \models \top \sqsubseteq \bot \Leftarrow \forall \mathcal{M} \subseteq \mathcal{O} : \mathcal{M} \models \top \sqsubseteq \bot$. In other words, if a module is inconsistent, then its parent ontology has to be inconsistent. Since $\mathcal{M} \subset \mathcal{O}$, by monotonicity it follows that $\mathcal{O} \models \top \sqsubseteq \bot$.

Now we show the $\Rightarrow$-Direction.

Claim: $\mathcal{O} \models \top \sqsubseteq \bot \Rightarrow \forall \mathcal{M} \subseteq \mathcal{O} : \mathcal{M} \models \top \sqsubseteq \bot$. In other words, if the ontology is inconsistent, each of its modules also has to be inconsistent. Since $\top, \bot \in \widetilde{\mathcal{M}}$ by definition it follows that $\forall \mathcal{M} \subseteq \mathcal{O} : \mathcal{M} \models \top \sqsubseteq \bot$.

If the $\mathcal{M}_\emptyset$ turns out to be non-empty, we perform an actual consistency check on the module. Since this module is hopefully small, the test should be faster than the one executed over the whole ontology. Note that some axioms are always non-local, most importantly ABox axioms and some of the ones involving nominals. As ABox assertions can pull in quite a large signature into the module for the empty signature (because ABox axioms are always non-local, and therefore part of every module), the method is perhaps less suitable for ontologies with ABoxes. We have conducted some preliminary evaluation for this approach, showing significant average speed-ups over the standard consistency check. Since any optimisation that is targeted purely at speeding up consistency checking may be of only limited use in practice, we did not include these results in this thesis.

3.3.6 Other Related Approaches

One related approach in First Order Logic is partition-based logical reasoning for first-order and propositional theories by Amir et al.[AM05]. This approach follows

a similar intuition as the decomposition-based reasoning we are concerned with. Similar to the Atomic Decomposition, their notion of a decomposition is that of a partition (i.e., no axiom appears more than once across the decomposition). The nodes in the decomposition are also sets of axioms, connected however by labelled edges that make explicit which elements in the signature are shared between the two connected nodes. Most of the approach is concerned with how messages are passed between the partitions. It is also motivated similarly to decomposition-based reasoning in our sense—by the question of how to reason effectively with multiple overlapping knowledge bases and generally of how to make reasoning (either locally or in a distributed way) over a set of axioms more efficient by using a partitioning. Similar to modular meta-reasoning frameworks, the authors discuss the possibility of integrating different reasoning procedures for each partition. An interesting side discussion is concerned with quality criteria for decompositions, for example related to the size of partition labels, the number of them and the size of their intersections—something that would be an interesting future work also with respect to the Atomic Decomposition.

The idea of partitioning ontologies in the way Amir et al. [AM05] proposed for First Order Logic has been (recently) applied successfully for ontologies with large ABoxes and tested for its utility to enable parallel reasoning [PGSH14]. While the approach claims to be sound and complete with respect to query answering, it is restricted to DL-Lite ontologies, and focuses on ABoxes.

An early mention of decomposition-based reasoning in Description Logics can be found in the work of Pham et al. [PLTS08] introducing the "overlap decomposing" technique. The work is based on Distributed Description Logics, defining mappings between names that co-occur between chunks in the decomposition. The authors saw the decomposition as a way to "minimize the execution time and storage space requirements". In another work, the same authors explore what they call *good* decompositions, as being efficient to reason over [PLTS07]. The algorithms used however were not based on modularity in our sense and the approach, based on the citation count, does not seem to have created any lasting interest.

In order to facilitate parallel reasoning for OWL ontologies, Wu et al. proposed a *merge-classification* algorithm for parallelising TBox classification [WH14], based on a divide and conquer strategy and a heuristic partitioning scheme to split the class graph of the ontology apart. The algorithm was also experimentally shown

to help avoid subsumption tests (at least over standard Enhanced Traversal) by providing improved exploitations of known subsumptions. However, the approach seems to be degrading performance quite often, and the principles by which it improves performance are not yet well understood.

There are other related approaches, for example the work of the Distributed Description Logic community, which is aimed at enabling reasoning between multiple ontologies connected by directional semantic mappings [Hom07], and Package-based Description Logics [BVSH09], a modular Description Logic language, but they are not directly applicable to monolithic reasoning and are not concerned with modules based on conservative extensions such as the ones we are using (Section 2.4). Examples of reasoners implemented by these communities are DRAGO [ST05], and DRAOn [DLZC13].

3.4 An Analytic Argument for Modular Reasoning

It is known that random subsets of an ontology $\mathcal{O}$ can be pathological, i.e. harder to classify than the whole ontology $\mathcal{O}$ [GPS12]. One obvious example for this is a missing disjointness axiom high up the hierarchy between two concepts A and B (in the subset) that makes testing the mutual subsumption of all children of A and B obsolete. Given the existence of such pathologically hard subsets, it is not immediate that module-based reasoning is going to be a straightforward, much less useful, optimisation. In this section we give an analytical argument for the use of modules in classification. As stated in the introduction, the overarching rationale for exploiting the modular structure of an ontology $\mathcal{O}$ for reasoning lies in its potential to improve reasoning performance by the following intuitions:

1. One or more modules in the decomposition can be processed by a delegate reasoner that is *more efficient* than the one that would be required to process the whole (Section 3.4.3).
2. The *traversal space* for subsumption tests *shrinks* significantly (Section 3.4.2).
3. *Subsumption testing gets easier* because there are fewer (irrelevant) axioms to sort through (Section 3.4.1).

We will discuss all three aspects in more depth in the following. However, these are not the only potential gains from modularity. Another example is a not yet well studied potential for parallelism. We will sketch the idea later in this

section.

3.4.1 Reducing Test Hardness

The time of a task to finish is the difference of a timestamp recorded just after the task terminates with the timestamp taken just before the task started, formally defined as follows:

Definition 3.1. *Given a task X, φ a function to obtain a timestamp, $\varphi(S)$ the timestamp taken when the task started and $\varphi(F)$ the timestamp taken when the task finished, we call* the time $T(X)$ *the difference between $\varphi(F)$ and $\varphi(S)$ $(\varphi(F) - \varphi(S))$.*

If it is clear from the context, we often omit the $T()$ function symbol when talking about the time it takes to execute a particular task (such as classification or subsumption test time). Note that this notion of time does not say anything about how the time was spent, in particular does it not reflect anything like memory consumption or concurrency. It is a purely empirical notion that just says: given some way of obtaining a timestamp (that must be specified, as we will see in the following chapter), how much time elapsed between the start and the end of the process.

Before we define subsumption test hardness in particular, we informally define the notion of experiment run as a single execution of a program (for example a reasoning task of $\mathcal{R}$ over $\mathcal{O}$) on a defined experiment machine (typically a server node or desktop computer). We can now define subsumption test hardness in terms of an aggregation function[4] as follows:

Definition 3.2. *Given an ontology $\mathcal{O}$, a reasoner $\mathcal{R}$, two concept names A and B, an experiment machine ϵ, an experiment run i, we call* individual subsumption test hardness *the time $ST(A, B, \mathcal{O}, \mathcal{R}, \epsilon, i)$ it takes $\mathcal{R}$ to decide whether $\mathcal{O} \models A \sqsubseteq B$ on ϵ in i. Given a set of experiment runs I and an aggregation function φ, we call* subsumption test hardness *the time $\varphi(\{\{ST(A, B, \mathcal{O}, \mathcal{R}, \epsilon, i) \mid i \in I\}\})$ aggregated across I by aggregation function φ.*

In other words, the hardness of a subsumption test between the concept names A and B is the time it takes to compute the subsumption in the context of an ontology $\mathcal{O}$, as computed by a reasoner $\mathcal{R}$, on an experiment machine ϵ,

[4] An aggregation function is typically one of mean, median, maximum or minimum.

aggregated across different runs by the aggregation function φ. For brevity, we always omit ϵ and φ, because they should be either inferable from the context (experimental design description) or are irrelevant for the argument at hand; we also sometimes omit $\mathcal{R}$, if it is clearly inferable from the context (or irrelevant). Irrelevant in this context means that the argument should hold for any *fixed* ϵ or φ (or $\mathcal{R}$).

Given a subsumption test $ST(A, B, \mathcal{O})$, it should be the case that, for every two modules $\mathcal{M}_1 \subset \mathcal{M}_2 \subseteq \mathcal{O}$ with $A, B \in \widetilde{\mathcal{M}_1}$, the hardness of $ST(A, B, \mathcal{O})$ should be (approximately) the same as $ST(A, B, \mathcal{M}_i)$, if we ignore the overhead involved in determining (ir)relevant axioms.[5]

The reason for this are the properties of locality-based modules: not only is every justification for an entailment part of every module for the signature of that entailment, but any module $\mathcal{M}$ with $\mathcal{M}_{sub} \subset \mathcal{M} \subseteq \mathcal{O}$ is a model-conservative extension of $\mathcal{M}_{sub}$. As a consequence, the space of possible models for $\mathcal{M}$ is more complex than that for $\mathcal{M}_{sub}$ in two ways: (1) since every model for $\mathcal{M}$ must contain a model for $\mathcal{M}_{sub}$, those models are larger than their embedded counterparts and (2) since models from $\mathcal{M}_{sub}$ could potentially be extended in multiple ways to models for $\mathcal{M}$, i.e., there is a 1-to-many relationship between models of $\mathcal{M}_{sub}$ and those of $\mathcal{M}$, the number of models for $\mathcal{M}$ is larger.[6] Figure 3.3 illustrates how, for a subsumption test $ST(A, B, \mathcal{O})$ between concepts A and B, model sizes might differ across modules. Thus, the task of finding a model or verifying that there are no models is, in principle, *at least as difficult* for $\mathcal{M}$ than for $\mathcal{M}_{sub}$ due to the larger space. An immediate consequence of the relationship between the models is that it cannot be the case that $\mathcal{M}_{sub}$ is consistent, while $\mathcal{M}$ is not (i.e. $\mathcal{M}$ has no models).

Let us consider the *case of a subsumption* where $\mathcal{M}_{sub} \models A \sqsubseteq B$ ($A, B \in \widetilde{\mathcal{M}_{sub}}$, $A \sqcap \neg B$ is not satisfiable w.r.t. $\mathcal{M}_{sub}$). We call $\mathcal{J}_{\mathcal{M}}^{A \sqsubseteq B}$ the set of all justifications for $A \sqsubseteq B$ in $\mathcal{M}$. Module properties ensure that $\mathcal{J}_{\mathcal{M}_{sub}}^{A \sqsubseteq B} = \mathcal{J}_{\mathcal{M}}^{A \sqsubseteq B}$, hence for all $\mathcal{J}_i \subseteq \mathcal{M}$, if $\mathcal{J}_i$ is a justification for $A \sqsubseteq B$ over $\mathcal{M}$ then $\mathcal{J}_i \subseteq \mathcal{M}_{sub}$, see above. Since all justifications are available in both modules, reasoning should not be harder in $\mathcal{M}_{sub}$: there can be no easier reason for $A \sqsubseteq B$ in $\mathcal{M}$ than in $\mathcal{M}_{sub}$.

Let us now consider the *case for a non-subsumption* where $\mathcal{M}_{sub} \not\models A \sqsubseteq B$

[5] Note that $\mathcal{O}$ (minus tautologies) is a module of itself.

[6] While normally adding non-redundant axioms shrinks the number of models of a theory, in this case we are also expanding the signature.

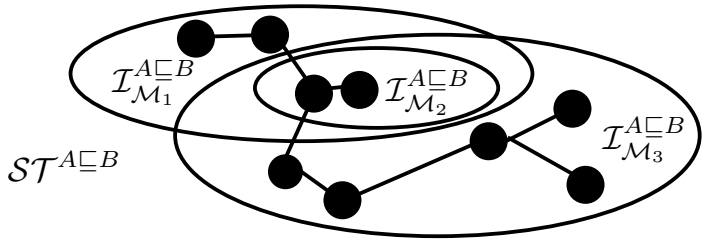


Figure 3.3: Example illustrating the different sizes of possible pseudo models when determining the satisfiability of $A \sqcap \neg B$. $\mathcal{I}^{A\sqsubseteq B}_{\mathcal{M}_i}$ is the model created for a particular module $\mathcal{M}_i$. Dots represent nodes in a pseudo model (instances) and the connecting lines the respective edges. Labels omitted for brevity.

($A, B \in \widetilde{\mathcal{M}_{sub}}$, $A \sqcap \neg B$ is satisfiable w.r.t. $\mathcal{M}_{sub}$). We call a model $\mathcal{I}$ of $\mathcal{M}$ with $(A \sqcap \neg B)^{\mathcal{I}} \neq \varnothing$ a counter-model of $\mathcal{M}$ and $A \sqsubseteq B$. Let $\mathcal{I} \models \mathcal{M}_{sub}$ with $e \in (A \sqcap \neg B)^{\mathcal{I}}$ a counter-model for the subsumption in $\mathcal{M}_{sub}$. From the considerations above we have that counter-models for $A \sqsubseteq B$ over $\mathcal{M}$ may not be less numerous or smaller than those over $\mathcal{M}_{sub}$. In other words, every counter-model in $\mathcal{M}$ has a smaller or equal-sized counter-model in $\mathcal{M}_{sub}$. If we consider the variants of locality-based modules, this consequence may be slightly more restricted. For example, consider the following example ontology $\mathcal{O}_{exp}$:

Example 3.1. *$\mathcal{O}_{exp}$:*

$\alpha_1 : A \sqcup X_1 \sqsubseteq X_2$

$\alpha_2 : X_3 \sqsubseteq A \sqcup X_4$

$\alpha_3 : A \sqsubseteq \exists R.\top$

The $\top$-module of $\Sigma_{sub} = \{X_3, X_4, A\}$ is just $\mathcal{M}_{sub} = \{\alpha_2\}$. A counter-model for $A \sqsubseteq \bot$ would merely consist of a single individual which is an instance of A. If we tested the same entailment with respect to $\mathcal{O}_{exp}$, we would at the very least add a non-empty interpretation of R, so the counter-model for $A \sqsubseteq \bot$ in $\mathcal{O}_{exp}$ is bigger than in $\mathcal{M}_{sub}$. The same does not hold for $\bot$-modules. Given a module $\mathcal{M}$ and its parent ontology $\mathcal{O}$ and the terms outside the module $\Sigma_x = \widetilde{\mathcal{O}} \setminus \widetilde{\mathcal{M}}$, then every $\alpha \in \mathcal{O} \setminus \mathcal{M}$ "is" a tautology when terms in Σ_x are replaced with $\bot$ ($\bot$-locality). From that follows, if $\mathcal{I}$ is a model of $\mathcal{M}$, the "empty" extension of $\mathcal{I}$ to terms in Σ_x ($\mathcal{I}'$) is a model of $\mathcal{O}$. However, building a tableau differs from the abstract notion of models. Highly optimised reasoners such as FaCT++ that are not aware of modularity sometimes grow models beyond the size strictly necessary. This can happen for example when dealing with general concept inclusions, which in turn depends on the many possible ways absorption is applied.

Note that the analytical argument presented in this section applies to tableau and hyper-tableau based classification algorithms and ignores the fact that many modern reasoners are hybrids that integrate multiple reasoning procedures based on different calculi[7] That means that axioms irrelevant for a particular subsumption to hold *can* make a difference in practice. For example, the removal of an inverse role that is irrelevant for determining the subsumption relation of two concepts enables the reasoner to resort to a cheaper reasoning procedure, therefore making the subsumption test potentially easier. Furthermore the *no-easier-justification* argument is rather theoretical: In practice, stochastic effects (both caused by algorithmic non-determinism and implementational aspects) can lead the algorithm into a harder space even when classifiying the sub-module.

Side Note: Hardness of Negative Subsumption Tests during Enhanced Traversal

Non-subsumptions outweigh subsumptions by far because the inferred class hierarchy is usually roughly tree-shaped. This is also reflected by the empirical observation that negative tests outweigh positive ones by a ratio of 40:1, occasionally more (see Section 6.4.1). A 100% (positive) subsumption ratio would mean that the class graph is a *complete graph*, i.e. all nodes $\mathfrak{V}$ in the graph are mutually connected (or, in other words, all $A \in \widetilde{\mathcal{O}}_C$ are equivalent). Since nodes in the class graph tend to have a much lower degree (in and out) then $|\mathfrak{V}|$, non-subsumptions outnumber subsumptions. This also has a consequence for the impact on performance. For (hyper-) tableau algorithms, negative tests differ from positive ones in one significant way: We only have to find one *counter-model* K for a subsumption $A \sqsubseteq B$, where a counter-model is a clash-free instantiation of the concept $A \sqcap \neg B$, while a subsumption needs to explore all non-deterministic branches of the tableau to show that there is no (clash-free) model at all. This means that a negative subsumptions should be easier than a positive one. As we will see in Chapter 6, this is not always the case. One reason for that may be the sheer amount of tests—there is simply a higher probability to hit a hard negative tests. Another explanation might be the efficiency of early clash detection optimisation such as lexical normalisation, simplification and synonym replacement in the case of a subsumption [TH06].

[7] For example Pellet and its internal $\mathcal{EL}$-classifier.

3.4.2 Subsumption Test Avoidance

Modules have a great potential to be used to prune the traversal space. This motivates the use of modules in particular for incremental reasoning. Figure 3.4 illustrates the consequence of the module property *subsumer-completeness* on the subsumption space, see Section 2.4 for a definition. A appears only in the signature of $\mathcal{M}_1$ and B only in $\widetilde{\mathcal{M}_2}$. Any locality-based $\bot$-module will contain, for each concept in its signature, all its superclasses. The consequence of this is that A and B cannot be in a subsumption relationship. We will illustrate how this principle can help us pruning the search space for subsumers considerably using two examples. Let us consider an ontology $\mathcal{O}$ and $A, B, C, D, E, F \in \widetilde{\mathcal{O}}$ with the following axioms:

$\alpha_1 : A \sqsubseteq B$

$\alpha_2 : C \sqsubseteq D$

$\alpha_3 : E \sqsubseteq F$.

A naive traversal algorithm (Section 2.3.2) would have to do 27 $(5 * 6 - 3)$ additional tests (between named classes) in order to fully classify $\mathcal{O}$. One possible decomposition of this ontology (based on the Atomic Decomposition) would result in three disconnected genuine modules, all of which are already fully classified: Subsumer-completeness guarantees that. In a slightly more realistic example, let us assume an ontology $\mathcal{O}$ with $|\widetilde{\mathcal{O}}| = n = 1000$, and a decomposition of $\mathcal{O}$ into two intersecting modules $\mathcal{M}_1, \mathcal{M}_2 \subseteq \mathcal{O}$ with $|\widetilde{\mathcal{M}_1}| = 600$ and $|\widetilde{\mathcal{M}_2}| = 600$. The number of tests theoretically (again naively) necessary to fully classify $\mathcal{O}$ is n^2, 1,000,000 tests. The number of necessary tests in order to fully classify the decomposed ontology is $|\widetilde{\mathcal{M}_1}|^2 + |\widetilde{\mathcal{M}_2}|^2$, 720,000: A reduction by 28%. In order for this calculation to work out for the much tighter n*log(n) upper bound, we have to assume a smaller overlap between the module signatures. If we assume $|\widetilde{\mathcal{M}_1}| = 530$ and $|\widetilde{\mathcal{M}_2}| = 530$, we have that the number of tests to classify $\mathcal{O}$ as a whole is 3000, and the two modules separately 2888—3.7% less. This is less impressive, but in practice modules will often be much smaller than half the ontology [VKP+13]. For example, if we assume 10 modules of signature size 110 each, we will get a total of 2245 test—a reduction by more than 25%. Note that this assumes a very naive modular reasoner which treats modules as blackboxes. The smaller the modules and their signature become, the greater they mitigate the quadratic effect n^2 (10 modules with signature sizes at 150 prune the search base by around 77% using the n^2 upper bound).

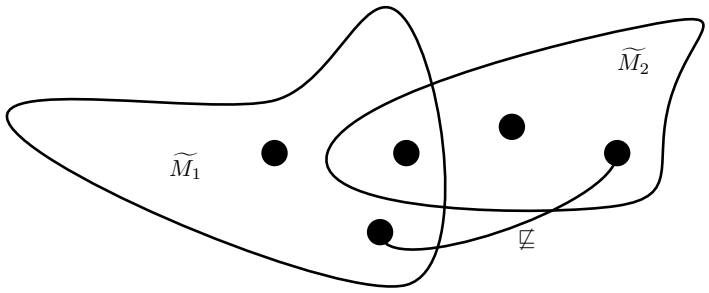


Figure 3.4: Two modules $\mathcal{M}_1$ and $\mathcal{M}_2$ of an ontology, dots representing names in the signature. Names that occur only in $\mathcal{M}_1$ cannot be subsumers of names in $\mathcal{M}_2$.

3.4.3 Integration of Efficient Delegate Reasoners

The other great potential of modularity, and also one of the core motivations of MORe, is the potential to mix and match *optimal reasoners* to modules in the decomposition. MORe for example extracts an as-large-as-possible $\mathcal{L}$-module of an ontology suitable for an efficient (in this case consequence-based) procedure, dispatches that to an efficient $\mathcal{L}$-reasoner available, and the remainder-module to an efficient OWL 2 reasoner. MORe does not, other than $\mathcal{EL}+$ to ELK, employ any further heuristics to make an *optimal* choice ("most efficient delegate"), and the delegate reasoners are (more or less) hard wired into the code. A truly optimal dispatching meta-reasoning framework would involve smart dispatching heuristics that, based on the particular properties of a module at hand (expressivity level, size, etc.), selects an optimal reasoner from a set of available ones. The smart heuristics could be obtained for example through a machine learning approach.

Note that modules and modular decompositions are a way of stitching different reasoning paradigms together: Whatever calculus is in some sense *optimal* for a particular module can be used to process it. We are *not* exploring *optimal delegates* as part of this thesis.

3.4.4 Modules for Parallelism

One obvious way to exploit the decomposition of an ontology is for *parallelism*. Parallelisation is one of the trend topics in recent reasoner development activities [MLH+15]. For example, Konclude [SLG14], a reasoning system for expressive DLs [PMG+15], and the most successful reasoner at recent ORE reasoner competitions (2014 and 2015) [PMG+15], employs parallelism on three levels of

its architecture. A modular decomposition could enable parallel reasoning by dispatching each module with a (suitable) delegate reasoner to a computational node in parallel, thus reducing the reasoning time of an ontology essentially to the time it takes to process the hardest module, plus some overhead for stitching the results together. However, these considerations remain in the realms of the hypothetical as there is, to our knowledge, no ongoing effort to realise modular decomposition-based parallel dispatch for OWL ontologies.

3.5 Limitations

There are some fundamental limitations on when and how modular reasoning is beneficial for OWL classification. There are two default cases where reasoning with a modular decomposition can have no benefits:

- The decomposition is of size 1 (module/chunk). Given an Atomic Decomposition, this can happen for example if (a) there is only a single root module or (b) there is only a single connected component. Given a MORe-decomposition, this is the case when $\mathcal{O}$ is already in $\mathcal{L}$ or $\mathcal{L}(\mathcal{O})$ contains nominals (or ABox axioms), which are not supported by the current implementation of the algorithm.
- The ontology is inconsistent. If the ontology is inconsistent, computing the entire decomposition is purely redundant, as everything is entailed by the inconsistent ontology (no expensive tests / traversal required). As outlined in Section 3.3.5, the module of the empty signature would be sufficient to determine consistency.

Apart from these two cases, there are a number of other cases that limit the applicability of modular reasoning in practice. Decomposition-based classification is interesting only if the overhead from dealing with the decomposition is outweighed by the performance gains induced by modularity, captured by the following definition:

Definition 3.3. *Given an ontology $\mathcal{O}$, decomposition-based classification can be said to be beneficial only if the following holds:*

$$T(Dec(\mathcal{O})) + \sum_{i=1}^{|\mathfrak{C}|}(T(Del(\mathcal{O}_i, \mathfrak{R})) + T(CL(\mathcal{O}_i))) + T(CR) < T(CL(\mathcal{O})),$$

where $\mathfrak{C}$ is the set of all chunks in the decomposition we are interested in, $\mathfrak{R}$ is the set of all reasoners we are interested in, Del (Delegation) is the process that decides which reasoner a subset should be dispatched to, $CL(\mathcal{O})$ is the process of classifying $\mathcal{O}$, and CR the combination of the results into a common structure.

In other words, decomposing the ontology, selecting a suitable delegate reasoner for each chunk, classifying each chunk and combining the results has to be faster than just classifying the ontology by a suitable (monolithic) reasoner. Note that these sub-processes of decomposition-based classification are in practice potentially intertwined. For example, result combination usually happens on the fly (for each module once it is fully classified), rather than in an isolated process afterwards.

This performance definition can be violated by current implementations in a number of ways, for example:

- The cost of computing the decomposition outweighs the benefits induced by subsumption test avoidance or easyfication of reasoning.
- The redundancy introduced by dealing with potentially overlapping chunks can outweigh the benign effects induced by search space pruning or easyfication of reasoning.
- Computing the classification of a single chunk in the decomposition can be harder (or almost as hard) as classifying the whole ontology at once.
- Allocating memory for a potentially large number of delegate reasoners can introduce a performance overhead, for example due to increased memory consumption.

Mapping the space of potential threats to the performance definition is one of the main goals of Chapter 7. In the following, we will present of some of the reasons why decomposition-based reasoning can be beneficial in practice.

3.5.1 Overhead

Decomposing an ontology (in the cases of Atomic Decomposition) can be done in polynomial (quadratic) time [DV13]. We have a linear number of module extractions (number of axioms * size of largest axiom signature), and the complexity of the locality check is linear in the size of the axiom. Insofar, modularity does not add to the computational complexity of classification, which is double exponential in the worst case for OWL 2 DL ontologies [HKS06] and polynomial for

ontologies in one of the three profiles [Kaz09].

It has been shown that, empirically, the Atomic Decomposition is extractable in a reasonable amount of time and that ontologies that timeout after 12 hours are typically large and contain a large number of individuals [HMP+14]. The overhead of computing the MORe-decomposition has not been, to our knowledge, studied systematically yet.

There is a range of strategies that can be employed to *mitigate the overhead* introduced by the decomposition of the ontology $\mathcal{O}$. We have to consider that, as discussed in Section 2.5.1, the Atomic Decomposition has only recently been implemented, and there is still a lot of room for optimisations of the algorithms. Moreover, as the authors of MORe point out, $\bot$-modules provide *too strong* guarantees for our purpose, as they not only preserve atomic subsumptions over the signature, but also *models* [RGH12, RKGH15]. The performance of decomposition and therefore decomposition-based reasoning will improve significantly if we find more efficient algorithms to extract more suitable modules, as long as they provide the necessary logical guarantees. One way to mitigate the overhead of decomposition entirely is to perform it *off-line*. The modular structure can be encoded directly in the distribution of the ontology or in the form of a simple module file that encodes axiom–module memberships. We have proposed a way to distribute decomposed ontologies as a set of modules connected by the owl:imports elsewhere [MP14c].

3.5.2 Module Hardness

One particular threat for reasoning with modules emanates from the observation that subsets of an ontology can occasionally be harder than the whole ontology [GPS12]. We have outlined in Section 3.4 the prima facie reasons for why the same should not hold true for modules (which are also subsets of an ontology). We have dedicated Chapter 5 to the question of whether this is—empirically—a *real threat* for module-based reasoning.

Note that stochastic effects caused by non-deterministic algorithms can, in principle, lead the algorithm to a *harder space* (i.e., a harder justification or a harder branch within the context of a justification), thus making a particular test appear harder not only across modules, but also across different runs (of the same module). We will show that state-of-the-art traversal algorithms are often impacted by stochastic effects in Chapter 6 and try to isolate other observations

as much as possible from randomly benign or detrimental effects caused by that stochasticity.

3.5.3 Redundancy

Redundancy is a special kind of overhead, insofar as it is *in principle avoidable*. Every sub-process executed by a monolithic reasoner classifying $\mathcal{O}$ can be potentially redundant in the classification of $\mathcal{O}$ by a decomposition-based reasoner. Examples of sub-processes that can be potentially redundant are:

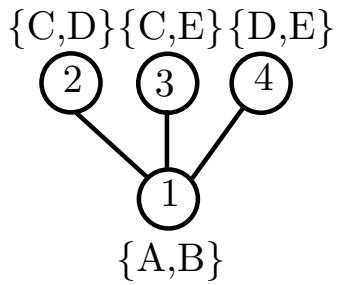
- consistency checking,
- pre-processing (normalisation, simplification, absorption) and
- subsumption testing.

This redundancy can be avoided if we introduce communication between delegates, i.e., by allowing the delegate reasoners to share some of their work. In contrast to this, overhead that is not caused by redundancy *cannot be avoided*: The decomposition, the dispatch decisions, the overhead in creating delegate reasoners (memory allocation) and the result combination can only be sped up, not avoided altogether. We have dedicated Chapter 7 to isolating the major sources of redundancy.

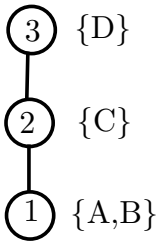
In the following, we will discuss the effect of the structure of the decomposition on redundancy. In Figure 3.5 we can see the modular structure of an ontology represented as its Atomic Decomposition. In case of maximal module classification, the worst case (for potential redundancy) occurs when the set of maximal modules RM depends on a common atom (of arbitrary size), and the label of each of the maximal atoms contains the entire set of remaining names minus one (Figure 3.5a). That way, to obtain the entire set of possible atomic subsumptions, each of the maximal module has to be classified, which introduces a large amount of redundant tests:

$$|RM| * (|\widetilde{\mathcal{O}}| - 1), \text{ where } |RM| > 1.$$

The worst case in terms of redundancy for classifying genuine modules is illustrated by the shape of the Atomic Decomposition graph depicted in Figure 3.5b. In order to obtain the complete set of entailed atomic subsumptions we have to classify each genuine module separately. It can easily be seen that classifying the genuine module of atom 3 is already equal to classifying $\mathcal{O}$, so that the rest just



(a) Worst case for maximal module classification.



(b) Worst case for genuine module classification.

Figure 3.5: The shape of the Atomic Decomposition in the respective worst cases.

adds work. In the worst case, the complexity would be (N_i being the size of the signature of the ith module)

$$|RM| * \sum_{i=1}^{|RM|} N_i(N_i - 1).$$

There are a number of cases where the shape of the decomposition does not introduce any redundancy. Apart from the trivial case of decompositions of size 1, this *can be* the case if the decomposition corresponds to a *partition* of $\mathcal{O}$, i.e. no axiom appears more than one chunk of the decomposition. It is trivially the case that, in such a situation, pre-processing algorithms that rewrite the input into an easier form would not do worse because every pre-processing optimisation would touch an axiom only once. The same does not apply for subsumption testing, as it is possible that two modules in $\mathcal{O}$ do not share any axioms, but share signature. For example, consider the following ontology $\mathcal{O}_{exp}$:

$\alpha_1 : A \sqsubseteq B$

$\alpha_2 : C \sqsubseteq \exists R.B$

There are four names in the ontologies signature $\widetilde{\mathcal{O}}_{exp} = \{A, B, C, R\}$). The $\bot$-modules for R and B are both empty. The $\bot$-module $\mathcal{M}_A$ for A is $\{\alpha_1\}$, and the module $\mathcal{M}_C$ for C is $\{\alpha_2\}$. Since an ordinary classification algorithm would always check class satisfiability, it would, in both cases, check whether B is satisfiable. Therefore, a decomposition being a partition does not immediately preclude subsumption test redundancy. However, if the reasoner was decomposition-aware, i.e., if was allowed to at least *peek* into the other modules, it could conclude from the mere fact that, given a decomposition $D_{\mathcal{O}}$ of $\mathcal{O}$, for

any $\mathcal{M}_i, \mathcal{M}_j \in D_\mathcal{O}$ with $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset$ and $\widetilde{\mathcal{M}_i} \cap \widetilde{\mathcal{M}_j} \neq \emptyset$ it follows that, for each entity $X \in \widetilde{\mathcal{M}_A} \cap \widetilde{\mathcal{M}_C}$, X is satisfiable. The reason for that is that by definition, modules preserve entailments over their signature and *all* justifications of an entailment are present in the module. Therefore, the sets of axioms that prove that $X \in \widetilde{\mathcal{M}_A} \cap \widetilde{\mathcal{M}_C}$ is unsatisfiable are present in all modules mentioning X, thereby violating the condition that $D_\mathcal{O}$ has to be a partitioning of $\mathcal{O}$. Apart from the unlikely case where no two modules in a decomposition share either axioms *or* signature (i.e., no intersecting models) we cannot currently prove, for a given case of a classification-preserving decomposition, whether *decomposition-awareness* precludes all kinds of redundancy.

There are various ways to *mitigate redundancy* for decomposition-based reasoning, but none of them are currently supported by those OWL reasoners that can be used as as delegate reasoners. The first obvious way would be to create a *reasoner with a modularity-sensitive calculus* as described in Section 3.1.1 that incorporates modular decompositions to enhance the current traversal algorithms (to avoid more tests) and serves satisfiability engines smaller sets of axioms to reduce the search space for building models. If we, however, hope to make use of *modular meta-reasoning frameworks*, we need other ways to enhance the communication between different delegate reasoners. Before we can hope to do that, we need to understand which sources of overhead and redundancy have the biggest impact, and therefore need mitigation. In Chapter 7, we will survey the sub-processes of reasoners that have impact on the overall classification time in depth.

3.6 Research Agenda

Many OWL ontologies still offer a challenge to state-of-the-art OWL reasoners. The quest for optimisations is ongoing. Modularity is employed by OWL reasoners in a variety of ways, for example for incremental reasoning (Section 3.3.4), identifying hot spots (Section 3.3.3) or full OWL ontology classification (Section 3.3). As the use of modularity necessarily imposes a computational overhead on any modularity-based reasoner optimisation, we need to improve our understanding of the interplay between the potential detrimental effects and the benign effects of modularity itself.

The assumption underlying all reasoning techniques involving modules is that

reasoning with modules is always easier then with the whole. We will introduce this assumption in Chapter 5 as the *benign module conjecture*. Only if we can show that this conjecture holds, or mostly holds, can we hope to benefit from modularity at all.

There are various ways in which modularity can help with reasoning (see Section 3.4). Two important effects are claiming to increase reasoner performance by reducing the input problem and pruning the traversal space for subsumption testing. We have presented a theory on how we expect modularity to affect subsumption test hardness 3.4.1 and traversal space pruning 3.4.2, but we have had no knowledge (1) whether these effects are significant in practice and (2) whether they make a difference on overall reasoning time. We have dedicated Chapter 6 to the question on how modularity affects subsumption test hardness during classification.

Ontology classification is probably the most important reasoning service in the area of ontology engineering. Even if we could be sure that reasoning with modules is always easier than reasoning with the whole, and subsumption tests become on average significantly faster (and fewer), we still cannot be sure that using them is always beneficial in practice, as dealing with modules necessarily comes with a computational overhead (for example module extraction). The overhead can be even more severe in the case of *decomposition-based* classification, as computing a decomposition may involve potentially numerous module extractions. In Chapter 7, we will study the performance profile of 5 decomposition-based reasoning approaches, investigating factors that contribute to the overall performance and subsumption test avoidance and redundancy.

Our systematic investigation into the viability of modules for use in OWL reasoners is only a *starting point*. MORe for example is primarily motivated by its use of optimal delegates, see Section 3.4.3. There is a large amount of interesting and well motivated investigation to be made in this direction. We cover delegate reasoners only to a very rudimentary extent. For example, we only really look *inside* OWL 2 DL reasoners, and largely ignore the internals of consequence-based ones. We will discuss some of these directions in Future Work, Section 8.2.

3.7 Summary

In this chapter, we have outlined the general motivation behind employing modules to improve classification performance and exposed important threats. In the remainder of this thesis, we will investigate some of the core issues that impact the use of modules for classification:

- The largest threat for modular reasoning emanates from the possibility that reasoning with modules might be harder than reasoning with their parent ontology. In Chapter 5 we investigate that threat in depth.
- One of our two core motivations to employ modularity is its beneficial effect on subsumption test hardness. In Chapter 6 we investigate the potential gain of hardness reduction and measure the effect of modules on subsumption test hardness during classification.
- In the last part of this thesis (Chapter 7), we analyse the classification performance of different decomposition-based reasoning approaches, in particular determining their effect on overhead and redundancy as well as test hardness and avoidance.

Chapter 4

Experimental Framework

One of the main contributions of this thesis concerns improvements to the methodological foundations of systematic experimentation with OWL ontologies. The core artefacts as a result of that effort are the OWL Experiment API, a lightweight framework that manages result reporting, time-outs, reasoner instantiation and more; the Reasoner Stage Benchmark, a methodology/library to measure the times a reasoner spends in its various stages and the duration of any subsumption test fired during classification and Katana, a framework to explore modularity-based classification (Figure 4.1). In this chapter, we will introduce these three components, and discuss some general aspects of our experimental setup, including randomisation, timing and the ontology corpus we used.

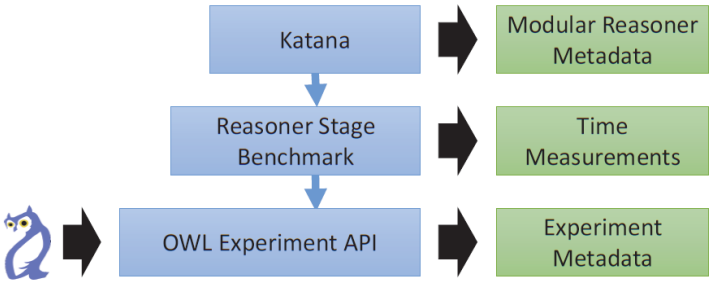


Figure 4.1: Overview of the experiment framework.

All pieces of software presented in the following were implemented drawing heavily on the OWL API [HB11]. We furthermore re-used some components from MORe [RGH12] ($\mathcal{L}$-module extraction) and borrowed from the implementation of Chainsaw [TP12a]. Analyses were performed in R [R C15], making heavy use of ggplot2 for plotting; other used packages include: data.table, xtable, plyr and reshape2.

4.1 The Reasoner Stage Benchmark

In the following we will describe the OWL Reasoner Stage Benchmark.

4.1.1 Overview

Most OWL reasoner benchmarks, especially those focused on classification, determine how long it takes the reasoner to execute the service for a given input. Few benchmarks distinguish between the different stages a typical OWL reasoner goes through, nor do they go as deep as measuring individual subsumption tests fired, or derivations rules triggered during classification. We propose the following model for traversal-subsumption test-based monolithic classification, distinguishing five stages for the process of classification. Note that this model is intended to capture tableau and hyper-tableau style reasoners such HermiT and FaCT++. Consequence-based reasoners in particular behave differently.

1. PP: Pre-processing (e.g. parsing, normalisation, absorption)
2. CC: Initial consistency check
3. PRP: Pre-traversal optimisations (e.g. leaf-node satisfiability)
4. ST: Traversal (e.g. Enhanced Traversal, Novel Approach, subsumption testing)
5. POP: Post-processing (e.g. generating the final inferred hierarchy)

The OWL reasoners we use in our work all follow that model, and we believe that most traversal/subsumption test-based OWL reasoners do—see also the more extensive discussion of reasoner architectures in Section 2.3. However, it is not specified what work a reasoner does exactly within each of the stages. A reasoner might defer the application of a particular optimisation another reasoner applies during pre-processing to a later stage (or the other way around). HermiT for example tests the satisfiability of the leaf-nodes of the known class graph during the PRP stage; no other reasoner triggers satisfiability tests (other than the initial consistency check) outside of the ST stage (traversal). This also has a consequence for the relative dominance of each stage for a reasoner: While HermiT has a very dominant PRP stage, for the other reasoners in our experiments this stage is often negligible (in terms of relative impact on overall classification time). The motivation for this fine grained view on the entire process of classification is as follows. First of all we get a sense of where the majority of the work is happening. If we consider the potential of modular reasoning, it is unlikely that there is much to be gained if the first stage is dominant (which it is in quite a number of cases). As we have outlined in Section 3.4, the main potential in utilising modularity lies in the reduction of subsumption test search space and the reduction of subsumption test hardness. That means that only the three Stages

(2-4) involving reasoning will be likely to benefit from modularisation. Without this fine grained view, we would miss a lot of explanatory potential for cases where modularity-based classification wins, or loses.

The second core aspect of the framework is the recording of subsumption tests. For a given reasoner $\mathcal{R}$ and a subsumption test $ST(A, B, \mathcal{O})$, we record the duration of the respective *call to the satisfiability engine* of the reasoner (or the respective equivalent). Timing subsumption tests is relevant in two ways for evaluating modular reasoning techniques: Firstly, from the number of subsumption tests, we can observe effects of search space pruning and redundancy. Secondly, from the duration of subsumption tests, we can observe test hardness. Another interesting phenomenon that can be analysed from the data of the Stage Benchmark is the test order, which can be a strong indicator for traversal non-determinism.

4.1.2 Implementation

From an implementation perspective, the framework currently has to be hard wired into the reasoner's source code. A single static Java class collects timestamps whenever a stage is entered or left, and whenever a subsumption test is conducted. While we did this by ourselves for the Java-based systems in our study, we collaborated with the developer of FaCT++ in order to extend the reasoner to merely flushing out textual information to a file that we then later parse back into our analysis framework. In order to minimise the effect on memory (especially when many subsumption tests are recorded), we use a buffered file writer to flush the data collected to a textfile. For the subsumption tests, we record start timestamp, end timestamp, result (true or false), the super-class and the sub-class. Each reasoner is referenced by an id based on the hashcode of its Java object, so that the framework can be used to collect data from modular reasoners involving multiple delegates. Time spent recording data is recorded as well, and can be factored out in the analysis. While the necessity to hard wire the Stage Benchmark into the reasoners source code creates some burden on the side of the developer, our experience is that the modifications required are reasonable. For example, the FaCT++ developer was able to make the necessary modifications in a reasonable time (less than 30 minutes). The modifications typically involve only small adjustments to the already existing metrics gathering facilities inside the reasoner.

4.2 Katana

A Framework For Evaluating Modularity-based Classification

Katana is a Java framework that allows exploring modular classification approaches. It allows mixing of different types of decompositions, module types and delegate reasoners with classification strategies.

4.2.1 Overview

The core aspects of our model of black-box, decomposition-based classification have been described in Section 3.2. Katana is an attempt to reflect that model and its aspects in a modular fashion.[1] The core motivation behind Katana is to make it possible to quickly implement and test new modular classification strategies and compare them to other approaches, including monolithic, MORe-like or Chainsaw-like approaches. Most aspects of modular classification, the decomposition, the module type, delegate reasoners (at least to some extent) and the classification strategy (responsible for determining the chunking, the order of the classification and delegate reasoner assignment) can be manipulated by the user. For example, a user might choose to explore a reasoning strategy based on the Atomic Decomposition that summarizes maximal modules that are in some way connected into chunks.

4.2.2 Implementation

The implementation of Katana is based on the classifier of Chainsaw [TP12a]. Earlier versions of Katana resembled Chainsaw, but only very few lines of code remained unchanged throughout development. We do acknowledge however that we built on the efforts around Chainsaw, and maintained helpful contact with the Chainsaw developers[2]. The main classes and their interdependencies can be seen in Figure 4.2. The KatanaReasoner class is the main class the user interacts with. It currently implements a minimal subset of the OWL API `OWLReasoner` interface, which makes it usable in the same way a normal reasoner would be

[1] Modular in the Java sense: extensible, re-usable.
[2] Dmitry Tsarkov and Ignazio Palmisano

used. The `KatanaReasoner` class contains a classifier, which manages the classification of the chunks provided by the `KatanaClassificationStrategy`. The `KatanaClassificationStrategy` summarises the modular decomposition (instantiating the `KatanaDecomposition`) into a set of chunks (`KatanaModule`) and then creates a list of `KatanaDelegation`(s), which are essentially assignments of a particular chunk to a delegate reasoner of choice. As a procedure, Katana works as follows (Procedure 4.1). Given an ontology:

Procedure 4.1. *Katana Workflow*

1. *User instantiates a decomposition of the ontology*
2. *User instantiates a classification strategy based on the decomposition*
3. *User instantiates the Katana reasoner with the classification strategy. The classification strategy summarises the decomposition into chunks, orders them and assigns them to delegate reasoners*
4. *User calls to compute the inferred class hierarchy*
5. *Katana determines consistency of the ontology*
6. *Katana classifies one chunk after the next according to order and delegate reasoner as determined by the classification strategy and merges the results by adding the inferences to a shared datastructure*

Sub-processes 1, 2 and 3 instantiate the reasoner. When the user asks for the class hierarchy to be computed, the reasoner first checks the consistency, and then starts classifying. The default classification strategy would simply feed one module after the next to the classifier, which would delegate the classification to a reasoner of choice. We allow Katana to be configured with a complete OWL 2 DL reasoner and an OWL 2 EL reasoner, and a strategy may choose which module to dispatch to the one or the other. As another example, the MORe classification would simply take the $\mathcal{EL}$-module computed as part of the MORe decomposition and dispatch it to the $\mathcal{EL}$ specific reasoner (usually ELK, but up to the user) and the remainder module would be dispatched to the full OWL 2 reasoner. After a chunk is classified, Katana incorporates the results into the inferred hierarchy and moves on to the next chunk.

In this thesis, we are looking at (and have implemented as part of Katana) five different classification strategies, which will be discussed in detail in Chapter 7. Two of the five strategies are used by available OWL reasoners, MORe and Chainsaw. The remaining three are variants of the approach employed by

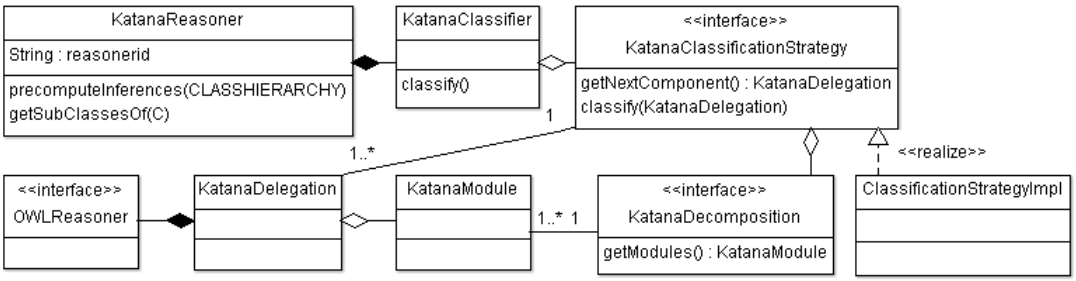


Figure 4.2: Most important Katana classes.

Chainsaw, addressing some of its obvious shortcomings. The MORe decomposition is implemented re-using components of the MORe implementation [RGH12]. Approaches based on the Atomic Decomposition are based in the OWL API tools implementation of the Atomic Decomposition.[3]

Katana as such is independent of reasoner implementations and versions: the only requirement is that it implements the OWL API `OWLReasoner` and `OWLReasonerFactory` interfaces in the typical fashion. It is available for download on the supporting materials website (see Section 4.8).

4.2.3 Katana Correctness

The algorithms underpinning Katana are proven to be sound, complete and terminating, and therefore correct. As implementations can be buggy, correctness needs to be verified experimentally. Up until today, we do not have any (feasible) means to verify the general correctness of a reasoner conclusively.[4] In order to ensure correctness, at least to a reasonable degree, we have implemented a set of tests which checks whether Katana produces the same classification as its primary delegate reasoner. The tests encompass all five modularity-based classification approaches implemented (see Chapter 7), Pellet, JFact and HermiT as delegate reasoners and 9 test ontologies, three of which are also part of our investigation in Chapter 7. At least three ontologies in the test set also contain unsatisfiable classes. As long as dispatch is restricted to Pellet, JFact and HermiT, all our tests pass.

[3] `https://github.com/owlcs/owlapitools`

[4] We have, however, started developing a justification-based method that attempts to isolate bugs by analysing disagreements between different reasoners over the class hierarchy [LMPS15].

4.3 OWL Experiment API

The OWL Experiment API is the layer in the experimental framework that manages general experiment configuration (metrics gathering, timeouts, reading and writing data).

4.3.1 Overview

Extensive OWL ontology and reasoner experimentation requires a lot of standard operations, such as loading and saving ontologies, creating reasoners and managing the measurements taken. Any serious empirical OWL researcher will eventually implement utility classes that factor out some of the recurring tasks. We went a step further and wrote a lightweight API that allows to rapidly implement new experiments.

4.3.2 Implementation

Figure 4.3 shows the three core (abstract) classes the user will mainly interact with. The `ExperimentRunner` manages the parameters taken from the command line, instantiates the experiment and manages the thread an experiment is run in using Java's single thread executors.[5] It also ensures the existence of the file the data is exported to and produces meaningful records if the experiment fails for some reason (such as timeout or bug). The `ExperimentRunner` executes the actual `ExperimentImpl`. We have provided an abstract Experiment class that manages aspects of the experiment such as writing the experiment results as comma separated values (CSV), gathering some default experimental metrics such as timestamps, experiment name, run id, and some high level file metrics, and managing file handles (to the ontology, the data directory and CSV files). The abstract `ReasonerExperiment` class extends that class by adding extra functionality to create reasoners, and gathers metrics about the reasoning. Along with these basic classes comes a wide range of utility classes for reading and writing files, extracting subsets of ontologies, generating normalised inferred hierarchies and many more.

[5] `http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html`

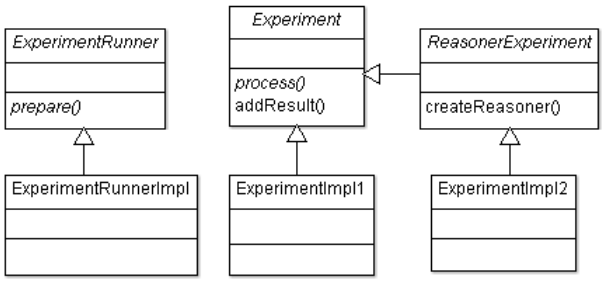


Figure 4.3: Most important OWL Experiment API classes.

The OWL Experiment API can be obtained from the supporting materials website (see Section 4.8). In order to set up a concrete experiment, the user creates a class that extends the `ExperimentRunner` and one of the abstract `Experiment` classes. The best way to start is to take an already implemented experiment such as `ClassificationTimeExperiment` and `ClassificationTimeExperimentRunne` and adapt it to the specific needs of the experiment at hand. The main advantage of using the API is that it takes care of experiment result formatting and metadata management (as described before), and provides some facilities to manage timeouts.

4.3.3 OWL API Classification

The following approach is used throughout Katana and the Stage Benchmark described earlier to classify an ontology.

```
public void process() {
        //...
        OWLReasoner r = createReasoner(o)\;
        r.precomputeInferences(InferenceType.CLASS_HIERARCHY);
        //...
}
```

4.4 Experimental Setup

Most experiments (typically the run of a classification of an ontology or a module) conducted as part of this thesis were run in the following fashion. A *command* is a program called on the commandline with a set of textual parameters, for

example:

```
java -Xms2G -Xmx12G -Djava.library.path=/Users/owl/libs/factpp/
    -DentityExpansionLimit=100000000 -jar ~/phd_cltime.jar ~/go.owl
    hermit ~/cltime/data/out.csv 1800000
```

1. A shell script iterates through the corpus directory, and writes all Java program executions (commands of the style described in the previous listing) that need to be executed as part of an experiment into a text file.
2. The order of the commands in that text file is randomized, and distributed across as many different text files as experiment machines are available.
3. The files are uploaded to the experiment machines.
4. The scripts are executed and every execution is logged in a text file.

We fully randomise the run order and evenly distribute the experiment run jobs across all available machines (of the same type) in order to reduce potential bias induced by run order. Despite efforts of reducing background processes kicking in during experiment runs (like deactivating automatic updates) we do not guarantee to account for all of them. An additional system process can harm execution times severely. Ideally, we would run experiments so often that such effects can be accounted for. Unfortunately, classification is a resource intensive task, and repeating experiment runs more than three times is usually unrealistic. Therefore, we sometimes apply a random-order, median-sampling approach, executing all experiments three times, scattered at random across all available machines, and then extract the run corresponding (or closest to) the median runtime of all equivalent runs.

Every single classification was done in a separate isolated virtual machine with fixed memory allocation size: 2GB minimum, 12GB maximum.

4.4.1 Timeout Management

Timeout management is one of the most painful aspects of ontology benchmarking. A proper timeout strategy is necessary mainly for two reasons. The need for dealing with timeouts in the first place is that we need to be able to interrupt reasoning (or module extraction) processes that potentially take days or even weeks to complete otherwise. The reason why we need to define a strategy is that the the implemented reasoners do not reliably throw exception on timeouts, and occasionally crash the virtual machine. Our time-out strategy involves four

different layers:

1. OWL API reasoner timeout (depends on the developers implementing the reasoner)
2. Java ExecutorService timeout (on Java's `Future` class)
3. Seperate timeout thread
4. System-level cron job to kill process

In most cases, the first layer works just fine. When the Timeout exception is thrown, we catch it from outside the process and collect some information about the failure. Sometimes, the reasoner gets stuck, but does not throw an exception. Then we rely on Java's ExecutorService to throw an exception and interrupt the process. Should this fail, the third layer guarantees to terminate the process, in all cases short of a JVM crash. If the JVM for some reason crashes (which may happen for example in conjunction with JNI-related libraries), we make sure that the process is killed from outside the system eventually, using cron-jobs and some custom shell scripts that terminate any experiment beyond the defined timeout.

4.4.2 Java Profiling

Profiling the execution of a process in Java 7 (version at time of this writing: Java 1.7.0_71-b14) is done in the same way across the experiments. For Chapter 7, resource limitations forced us to outsource one experiment to Amazons EC2 running Java 1.8.0_45. The following considerations apply to both Java versions.

There are two important decisions to be made when measuring execution time: (1) Are we interested in user time or CPU time? (2) What resolution do we need (minutes, milliseconds, nano seconds)? Neither question has a clear answer. For question one, we have to choose between taking CPU time, for example using Java's ThreadMXBean.getCpuTime(), or taking a System timestamp before and after a given process. The latter is also referred to as *wall-clock* time, representing the time a user is waiting for the task to complete, while CPU time reflects the time the CPU is running application code or operating system code on behalf of the application. The strongest argument against wall-clock time (WCT) is its considerable sensitivity to system level background processes. Especially on Windows machines it is not advisable to use WCT if one is interested in very precise measurements of short processes due to inconsistent resource distributions when the operating system changes its priorities.[6]

[6] `http://nadeausoftware.com/articles/2008/03/java_tip_how_get_cpu_and_user_`

The second decision to be made concerns the resolution of the time taken. In Java, we can for example choose between `System.currentTimeMillis()` and `System.nanoTime()`, the former having a resolution of a thousandth of a second, and the latter, at least in theory, a billionth of a second. While System.nanoTime() provides nanosecond *precision*, it does not provide nanosecond *accuracy*. Depending on the operating system, it usually is accurate to the microsecond, without providing any particular guarantees. There is a lot of ongoing debate about whether System.nanoTime() should be used at all. The main reason against it is that its implementation is system dependent. Depending on the hardware and OS used, it can either return elapsed CPU time based on the programmable-interval-timer (PIT), the ACPI power management timer (PMT), the CPU-level timestamp-counter (TSC) or (on very old systems) a value based on System.currentTimeMillis(). In general, System.nanoTime() does not relate to any real clock. It measures the elapsed time in nanoseconds from a fixed but arbitrary start time, and can therefore be used exclusively for measuring the elapsed time of a process. The second problem with it is that, depending on the system architecture, the behaviour of System.nanoTime() varies in programs that make use of multiple cores. This is because it usually depends on CPU time, and not all operating system synchronise CPU time between different cores. On the other hand, it has two advantages over System.currentTimeMillis() beyond the increased resolution: It is robust against irregular lapses in wall clock time (for example, if the clock is adjusted for some reason, like winter and summer time) and it is more robust against other system level activity because other processes would, on a multi-core architecture, be dealt with by another core. Despite its shortcomings, we decided to use it to accommodate for the fact that many individual subsumption tests are, in terms of execution time, in the realms of microseconds. Using System.nanoTime() is clearly debatable, and future work should involve comparing results obtained with System.nanoTime(), System.currentTimeMillis() and CPU time. The use of System.nanoTime() over System.currentMillis() has been strongly encouraged by IBM[7] [Boy08] and Oracle[8], despite the many passionate voices against it that point out, for example, its dependence on hardware.

Side note on timings: We considered the importance of taking a timestamp very late in the course of this PhD, relying on previous work that made use of

time_benchmarking

[7] `http://www.ibm.com/developerworks/library/j-benchmark1/`

[8] `https://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks`

System.nanoTime(). Given the multi-threaded nature of many state-of-the-art reasoners, we do believe that experiments should be conducted to determine the best way to measure reasoning performance, also with respect to the operating system. This would include questions such as accuracy of the measurement, wall-clock vs CPU time, variation between operating systems and Java versions, and the exact overhead of the timing. The four OWL reasoners that we included in our experiments do not make use of multiple cores. While we have not systematically investigated this, we gain some confidence from the fact that (1) no duration calculated is ever negative and (2) no ratio we calculated between nested sub-processes is ever beyond the range of 0-100%, for example the ratio of sum of subsumption test to the overall classification time.

The following listing shows an example measurement using System.nanoTime().

```
public void process() {
        //...
        long start = System.nanoTime();
 reasoner.precomputeInferences(InferenceType.CLASS_HIERARCHY);
        long end = System.nanoTime();
        //...
}
```

4.4.3 Experiment Machines

In the following we list the experiment machines used in our experiments. The Mac Mini cluster was picked for convenience: we needed access to a cluster for many months in total and we needed to be able to control system level processes (mainly to prevent them from kicking in an skewing the results). The second cluster (Amazon EC2) was also fully configurable, but could become quite costly for long running experiments. We do not believe that the particular operating system biases the results in any systematic way (for example by having longer subsumption test times, while having shorter pre-processing times), but this fundamental assumption should be substantiated in future experiments.

Experiment Machine 4.1. *Mac-mini: A set of four (equivalent) Mac Minis with Mac OS X Lion 10 (64 bit), 16 GB RAM and 2.7 GHz Intel Core i764-bit.*

Experiment Machine 4.2. *Amazon EC2, r3.large: Ubuntu 14.04.2 LTS, Memory: 15.25 GB, Intel Xeon E5-2670 v2 (Ivy Bridge) Processors @ 2.5 GHz*

X 10

4.5 Thesis: Metrics

There are two measures we make frequent use of in our analysis:

- Fold change (FC): describes how much a quantity changes from an initial to a final value.
- Coefficient of variation (COV): measure of dispersion, defined as the ratio of the standard deviation to the mean.

The *fold change* is used to emphasise how much a quantity changed from one situation to another. Given two time measurements m_1 and m_2, it is calculated by division, $\frac{m_1}{m_2}$, describing the change from m_1 to m_2. In order to make negative and positive changes comparable in terms of magnitude, we can calculate the *normalised fold change* (FC) by replacing changes lower than 1 by the negative of its inverse as follows: if $m_2 \geq m_1$, $FC(m_1, m_2) = \frac{m_1}{m_2}$, else $FC(m_1, m_2) = -(\frac{m_2}{m_1})$. The problem with this normalisation is that the range between -1 and 1 is not within the range of the metric. As we use this metric only for describing the change, and no further computation, we occasionally redefine the normalised fold change (RFC) by "collapsing" the region between -1 and 1: if $m_2 \geq m_1$, $RFC(m_1, m_2) = \frac{m_1}{m_2} - 1$, else $RFC(m_1, m_2) = -(\frac{m_2}{m_1}) + 1$ (redefined normalised fold change). The fold change can be converted into a *percentage change* by using the formula $(n-1)*100\%$, where n is the fold change (or normalised fold change). For the *redefined normalised fold change*, the formula is simply $n * 100\%$.

The *coefficient of variation*, sometimes referred to as relative standard deviation, is a measure of dispersion that we use to analyse variation across measurements that are not on the same scale (for example seconds and milliseconds). Given a list of measurements M (of the same kind) it is computed as follows: $\frac{sd(M)}{mean(M)}$, where $sd(M)$ is the standard deviation of M and $mean(M)$ is the mean of M. For example, given three independent measurements of a subsumption test $ST(A, B, \mathcal{O})$ (in milliseconds) $\{100, 120, 90\}$, and three independent measurements of a subsumption test $ST(C, D, \mathcal{O})$ $\{1000, 1500, 950\}$, we can compute the COV for both: $COV(ST(A, B, \mathcal{O})) = \frac{15.28}{103.33} = 0.148$ and $COV(ST(C, D, \mathcal{O}))$ $= \frac{304.13}{1150} = 0.265$. $COV(ST(A, B, \mathcal{O}))$ and $COV(ST(C, D, \mathcal{O}))$ are now comparable, and we can see that $COV(ST(C, D, \mathcal{O}))$ has a larger degree of variation than $COV(ST(A, B, \mathcal{O}))$. The coefficient of variation is usually reported as a

percentage, simply by multiplying it with 100. For example, given the above coefficient of variation $COV(ST(A, B, \mathcal{O})) = 0.148$, we read "measurements for $ST(A, B, \mathcal{O})$ varied by 14.8%" (0.148*100).

4.6 Reasoner Benchmarking

As described in Chapter 2, reasoners are complex pieces of software, for example, due to their complex and possibly intertwining optimisations and implementational particularities. Some reasoners, such as FaCT++, use up to 8 different absorption techniques during pre-processing at once [TH06]. This precludes their availability to formal investigations of reasoning hardness (see Section 1.3). In order to assess the viability of reasoning procedures or particular optimisations, we are left with the empirical toolbox. The process of measuring aspects of OWL reasoning performance is called *reasoner benchmarking*.

The pressure on showing the viability of reasoning techniques has increased, and is to increase more due to reasoning being considered for industrial solutions, like Pellet for the enterprise graph database Stardog[9] or in the context of Optique [GCH+13]. To date, the empirical methodological toolkit of the community is still preliminary, compared to other disciplines such as Software Engineering [SCVJ08]. In the following, we will summarise some benchmarking activities of the community, illustrate some of the shortcomings of our methodologies and summarize some of our suggestions to improve the sate.

4.6.1 Brief Survey of Reasoner Benchmarks

Attempts to understand DL reasoning performance are, up until today, rarely systematic and independent. The recently established ORE reasoner competition tries to establish the methodological foundations for more reliable comparisons between different reasoners and across a range of different reasoning services [GBJR+13, PMG+15]. Most benchmarking activities are second class citizens in system description papers or evaluations of particular optimisations.[10] However, there have been a few attempts to conduct systematic, independent

[9] `http://stardog.com/`

[10] A claim based on the authors experience that should really be substantiated in the form of a systematic review.

evaluations of reasoners. We will discuss some of them in the following, and leave a comprehensive review for future work.

OWL reasoner benchmarks have been designed for various purposes, for example (and most prominently) for guiding end-users to select appropriate reasoners for their problem [DCtTdK11, GBJR+13], understanding reasoning or the state of reasoning in general [GMPS13] and to explore the viability of particular optimisations for reasoning. Early proposals for DL reasoner benchmarking, long before the dawn of OWL, remained coarse grained [HPS98], but covered synthetic as well as real problems. Tempich et al. [TV03] perform statistical analysis on real ontology corpora to inform the generation of synthetic reasoning problems (written in DAML, a precursor of OWL). They also describe different reasoning services to be tested and basic metrics any benchmark should cover. Gardiner et al. [GTH06] proposed an automated framework for testing OWL reasoners that covered more aspects of the classification process than overall classification time, such as consistency checking and satisfiability. They do not however observe these tasks as part of the classification process. Like most other frameworks that cover correctness, the authors suggest majority voting over the class hierarchy as a suitable approximation. Weithöner et al. [WLL+07] provide an early example of trying to understand the requirements for systematic OWL reasoner benchmarking to make them more useful for application developers. Babik et al. [BH08] also emphasise the lack of frameworks for automated testing of OWL ontologies and propose a methodology that covers both ABox and TBox reasoning involving real-world ontologies and another early mention of majority-voting to determine reasoner correctness. LUBM is a benchmark for OWL knowledge base systems involving synthetic ontologies [GPH05]. LUBM's major limitation is that the artificial problems it generates stand in an unknown relationship to real-world problems. Despite that, based on citation count, LUBM is the most widely used benchmark for OWL reasoners. However, it is used almost exclusively for evaluations of query answering approaches, and the ontology itself is, at least in its basic version, very simple, i.e. expressed in DL-Lite. Ma et al. [MYQ+06] extend LUBM to more expressive TBoxes, but is again aimed at query answering rather than TBox reasoning. Dentler et al. [DCtTdK11] conduct a principled investigation to identify suitable criteria for choosing an appropriate reasoner for an OWL 2 EL ontology. The authors conduct one of the very few independent and systematic surveys of classification performance, which served as inspiration

for parts of our work.

No benchmarking solution is perfect. For example, key properties leading to the dataset selection are unknown, which makes dataset selection (and therefore the experiment) not reproducible. Even carefully executed benchmarks such as those by Dentler et al. [DCtTdK11] usually cherry pick a set of *somehow* relevant ontologies, where *somehow* is subject to the author's intuition. A more principled selection would be, for example: "all OWL 2 EL ontologies on BioPortal", or "the most frequently cited bio-health ontologies". It is hard to make substantive conclusions, if we cannot even determine whether a particular set of ontologies is somewhat comparable to the one used in an experiment (due to the lack of explicit selection criteria). Few works sample from existing corpora or directly from the web, and only Gonçalves et al. [GMPS13], to the best of our knowledge, deal with corpora larger than 500 ontologies. In practice, the current de facto gold-standard corpus for ontology experimentation is BioPortal [NSW+09], which also provides a well designed infrastructure to obtain an interesting range of biomedical ontologies programatically. We are using a snapshot of BioPortal in this work. As far as we know, no benchmark to date has investigated *subsumption testing during classification* across reasoners in a principled manner. However, various benchmarks have been designed to investigate the effect of certain optimisations on subsumption test avoidance [GHM+12].

4.6.2 The Quest for the "Ultimate" Dataset

As mentioned before, one of the core shortcomings of current reasoner evaluation methodologies is the absence of an agreed upon set of problems (i.e. ontologies, queries, services). In an ideal scenario, an evaluation would yield an outcome such as "Approach A is more efficient than approach B for function X for all cases P (with criterion C)", where X is a reasoning service such as classification, A and B are reasoning algorithms or particular optimisations, P is a set of representative problems (ontologies, queries) and C an optional set of criteria that restrict the problem-space in a meaningful way. An author of an optimisation or a novel reasoning procedure typically picks a set of problems P by hand, which has various and often severe consequences to the generalisability of the results (so called cherry-picking).[11] For the case of classification benchmarking,

[11] A notable exception for the lack of rigour in empirical evaluations is the work by Steigmiller et al. [SLG14]

the problem-space we are interested in are typically a subset of all OWL 2 DL ontologies: OWL 2 DL (EL, QL, etc.) ontologies that are likely to resemble the ones used in practice.[12] In the absence of a dataset that reflects the set of all ontologies in some agreed upon fashion, let alone some indication whether ontologies are actually used in practice or not, this creates a considerable burden on the developer: Gathering and pre-processing ontologies in a principled manner is a very hard and time-consuming activity. Throughout the years, some corpora such as Tones[13], BioPortal [NSW+09], the Oxford Ontology Library[14] and the Manchester-based crawl-based corpus MOWLCorp [MBP13b] emerged, but all of them fail to be convincing samples of the entire problem-space.

MOWLCorp was created as part of the work of this thesis, but in the end failed to exhibit some of the desired properties of a benchmarking corpus, namely to have a roughly normally distributed spread of problem *hardness*. It however had a property that none of the other corpora could claim: the (almost) absence of selection bias. Both Oxford and Tones were put together for a very particular purpose: Tool testing. Both corpora contain a mix of toy ontologies and real biomedical ones. Neither were ever updated in the face of new ontologies or ontology versions (and therefore potentially new modelling patterns that might cause problems for reasoners). MOWLCorp was based on an ongoing crawl, which made the creation of *snapshots* possible, each of which potentially contained new ontologies and updated ones. Details of the methodology can be found elsewhere [MBP13b, MTPS14], later refined and extended for the ORE reasoner competition [PMG+15].[15] In the end, we decided that for our purposes, BioPortal [NSW+09] would be the repository of choice. This selection will be motivated further in the following section.

4.6.3 Thesis Dataset: BioPortal

BioPortal [NSW+09, SAMN13] is a repository for biomedical ontologies. Labelling it a *corpus* of ontologies is actually a misnomer, as BioPortal is a complex software for visualising, versioning and accessing ontologies, along with services to automatically annotate resources and search for terms. We are primarily interested in BioPortal because of the bio-health ontologies hosted as part of its

[12] Some reasoners such as Snorocket tune towards specific ontologies.
[13] `https://www.w3.org/2001/sw/wiki/TONES`
[14] `http://www.cs.ox.ac.uk/isg/ontologies/`
[15] `http://bit.ly/1J2X3Vi`

repository. Many of the ontologies as part of BioPortal are built to be used by domain experts and information systems, and the both the end-users and engineers are important stakeholders for OWL reasoners. We keep a copy of all versions of all ontologies publicly available on BioPortal (7897 files in January 2015) through its REST Services.[16] We call a *snapshot of BioPortal* the set of latest versions of all ontologies accessible through these web services. Due to the fluctuating availability of imports and the quirks of different OWL API versions with respect to different OWL and OWL-compatible syntaxes we create a working BioPortal corpus by serialising every ontology in a snapshot into *OWL/XML* with *merged imports closure*. A minimum amount of repair is applied to ensure that trivial (syntactic) violations do not impair DL-ness, such as injecting missing declarations and removing empty language constructs. We have presented a more detailed discussion about our repair strategy and an extensive survey of OWL use beyond the expressivity bounds of OWL 2 DL elsewhere [MP14b]. Our snapshot contains 330 non-empty ontologies. In Figure 4.4 the division of ontologies into their respective profile categories can be seen. As discussed in Section 2.2 the three OWL 2 DL Profiles EL, QL and RL intersect. In order to keep the bins exclusive (every ontology only belongs to a single bin) we distinguish between 7 sub-bins for the profiles. Each bin label should be read as "including *label*", "excluding other profiles". For example EL+RL means "including *EL+RL*, but excluding QL". We also distinguish between *merely hierarchical RDFS* (RDFS(-)) and *supra-hierarchical RDFS* (RDFS[17]). There are 57 RDFS ontologies in total, 53 of which are RDFS(-) and 4 out of which are full RDFS. RDFS(-) are those RDFS ontologies that contain only subclass axioms (57 ontologies contain subclass axioms), sub-property axioms (5 contain sub-property axioms), class assertions (4), data property assertions (2) and object property assertions (2). Supra-hierarchical RDFS occurs rather rarely and furthermore includes object property domain (4 ontologies) and range (2) and data property domain (3) and range (2) restrictions. 48 RDFS ontologies can be found in the intersection of the three polynomial OWL profiles (EL+QL+RL bin) (Figure 4.4), with most of them (46) being represented in RDFS(-). 6 ontologies in the intersection of EL and QL fall under RDFS(-), and 1 ontology in the RL bin.[18] Only 2 RDFS

[16] `http://data.bioontology.org/documentation`

[17] The method for RDFS detection can be reviewed in the Appendix, section A.2

[18] The difference between RL and EL+QL is that RL allows for anonymous individuals [MGH+12].

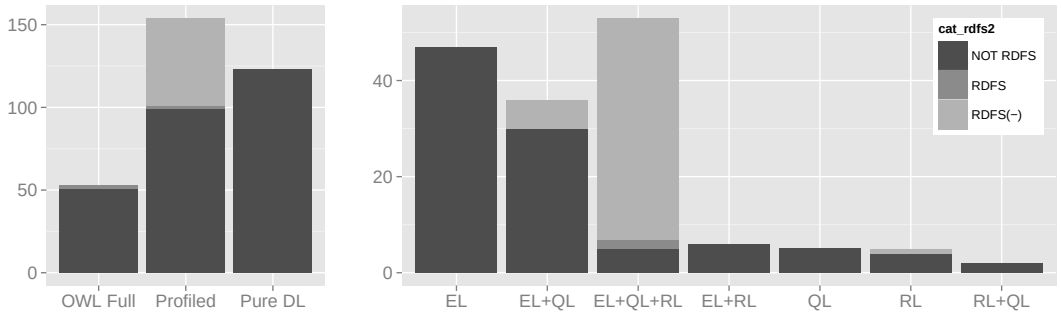


Figure 4.4: OWL 2 Profiles. Right: Detailed analysis of *Profiled* category on left chart. Y: Number of ontologies; X: Profile

ontologies do not fall under any of the OWL 2 profiles, both OWL Full. Another observation is that around 43% of the BioPortal ontologies in our corpus are in OWL 2 EL, i.e. they could be dealt with by an efficient OWL 2 EL reasoner such as ELK. This is particularly important to understand when evaluating a modular reasoner that follows a MORe-like strategy that depends on first dividing the ontology into an $\mathcal{EL}$ and non-$\mathcal{EL}$ part. By default, it is very unlikely that there is much to be gained from classifying one of the 141 ontologies under the EL profile with MORe. If we also consider the 53 ontologies in OWL Full, we have to admit that a MORe-type approach might not be applicable to almost 60% of the ontologies in BioPortal.[19]

123 ontologies in the corpus (around 37%) fall under OWL 2 DL, excluding the 153 that fall under one of the profiles. Only 7 of these ontologies involve modelling across the entire range of $\mathcal{SROIQ}(\mathcal{D})$, a further 5 correspond to the expressivity of $\mathcal{SROIQ}$ without datatypes. The main thing to learn from this is that more than a third of the ontologies in the corpus are fairly expressive. If one accepts BioPortal as a proxy of the population of OWL 2 DL ontologies, this creates a serious necessity to support the development of reasoning techniques for very expressive ontologies.

In Figures 4.5 we can see that most ontologies in the corpus are predominantly TBoxes. More than 36% of the ontologies have between 100 and 1,000 axioms and another 36% between 1,000 and 10,000, compared to only around 18% larger than that, and around 9% smaller. This more or less even distribution across the relevant size bins suggests that reasoning procedures intended for general

[19]Probably more, if we take into account implementation limitations such as individuals and nominals and the fact that 37 ontologies under DL contain nominals.

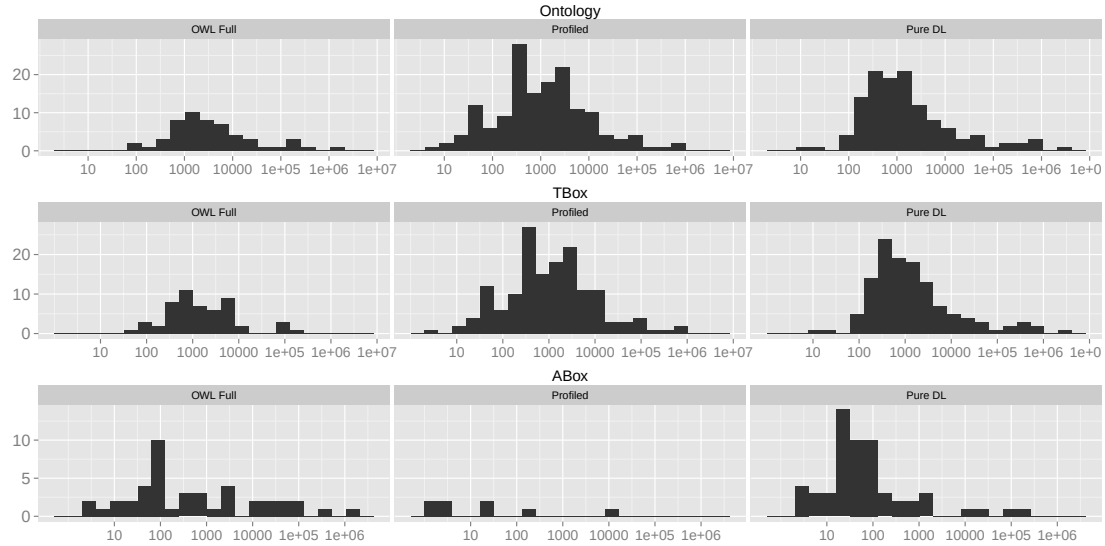


Figure 4.5: Histogram of axiom counts across the corpus. x-axis: number of axioms, y-axis: number of ontologies. Profiled ontologies are those that are either OWL 2 EL, QL or RL, and Pure DL ontologies are those that fall under OWL 2 DL, but not under one of the profiles.

purpose reasoners should not focus on ontologies of particular sizes. Claims that a specifically tailored reasoner might be able to deal with very large ontologies should always be weighed against the knowledge that many ontologies are not necessarily large, but perhaps smaller and very hard. More information about the corpus can be found online.[20]

In the remainder of this thesis, we group ontologies by size bins. Ontologies with less than one axiom are considered empty, with 1-9 axioms *very small*, with 10-99 axioms *small*, 100-999 *medium*, 1000-9999 *large*, 10,000-100,000 *very large* and with more than 100,000 logical axioms *huge*. These bins do not aim to reflect the actual distribution of ontology sizes. The main purpose of introducing them is to make some parts of the analysis easier to understand; therefore we chose the bin ranges in a way that is easily memorable by the reader.

[20] `http://mowlrepo.cs.manchester.ac.uk/datasets/bioportal/`

4.7 OWL Reasoners

4.7.1 Overview of the DL Reasoner Landscape

Apart from contributing some insights into the viability of modular reasoning techniques, we have developed an extensive toolkit for reasoner evaluations, primarily focusing on dataset creation and improved benchmarking. Some of these tools have become a regular part of the ORE framework, primarily used for the annual ORE Reasoner Competition [PMG+15]. As a by-product of our *ORE recruitment strategy* which aimed at involving as many up-to date reasoning systems to the competition as possible, we conducted a survey on stand-alone, current reasoners supporting reasoning with OWL or a fragment of it [MLH+15]. This survey is relevant for research into modular reasoning in two ways. Firstly, the survey forms the starting point of a reasoner index (listing). One of the main motivations behind employing modularity is to utilise it for *optimal dispatch*, i.e. processing subsets of the ontology by the best suited delegate reasoner.[21] An accessible index of reasoning system is needed in order to be able to select the best suited reasoner for any given problem. Secondly, we were particularly interested in the question of how many reasoners were utilising modularity, and in what ways. We mention some relevant outcomes of the survey in the course of this thesis where appropriate; details can be found elsewhere [MLH+15]. The complete listing of the reasoners can be accessed online [SM15]. The systems that participated in the survey can be found in the appendix, Table A.2.

4.7.2 Reasoners in this Thesis

For all our experiments, we use four OWL reasoners that implement the OWL API interface: HermiT 1.3.8, Pellet 2.3.1, JFact 1.2.3 and FaCT++ 1.6.3. All four are among the most heavily used reasoners[22] for OWL 2 DL. HermiT uses a hyper-tableau approach, while the other three reasoners employ standard Enhanced Traversal / tableau-based techniques. Despite the differences of hyper-tableau and normal tableau, both fit under the model outlined in Section 4.1. For

[21] For example in MORe: ELK for $\mathcal{EL}$ and HermiT for remainder.

[22] This has never been formally verified. Indications of usage are: default shipping with Protégé, citation counts of system description papers and download counts.

the remainder of the thesis, a subsumption test is a test either triggered by a Enhanced Traversal / tableau reasoner *or* a hyper-tableau reasoner. The reasoners have been modified for the benchmark: When a subsumption test is conducted, the start and end timestamps, the sub and super class under consideration and the result of the test are recorded, see Section 4.1.

As of 2014, the new flagship reasoner of the DL community is Konclude [SLG14], and every thesis around OWL reasoners must have a good reason for excluding it from its experiments. In our case, Konclude had to be excluded for three main reasons. Firstly, at the time of running the experiments, there was no really convenient way to interacting with Konclude through the OWL API. The only way was through OWLlink[23], which required the user to first start a Konclude server instance outside of the virtual machine the experiment was run in. This way of interacting with the reasoner created some (often parsing related) bottlenecks, and inconveniences in terms of experimental setup (killing the server after each classification, waiting for the operating system to free the port it was run on, and more). Secondly, implementing the Reasoner Stage Benchmark would have required effort from the Konclude developers. As we were only just developing the Stage Benchmark, it was more convenient to interact with developers more local to us (FaCT++). Lastly, and most importantly, our time management was at the time of writing not robust for parallel implementations such as the ones found in Konclude. It is likely for example that the use of System.nanoTime() is problematic when benchmarking Konclude.

While we can use this approach to compare results for each reasoner, interpretation of comparisons between reasoners might be misleading due to implementation particularities. For example, some reasoners might apply graph-based methods to determine subsumptions up-front, and other might have them more tightly intertwined with their (hyper-) tableau engine. In order to choose where exactly to measure, we either asked the developers directly (JFact, FaCT++) or were guided by benchmarking code already present (progress monitors for debugging in HermiT and Pellet). The exact points of measure can be taken from Table 4.1. Because we are interested in real life behaviour, we allowed the reasoner to fall into any internal state it normally would, like the deterministic part of HermiT for Horn-SHIQ or Pellets internal EL-Reasoner. That said, we do not

[23] `http://www.owllink.org/`

claim to time `all` subsumptions a reasoner determines, because they are also determined by nested consequence-based approaches or during pre-processing using structural approaches. We are confident that we capture all tests determined by actual calls to the tableau engine *during traversal*.

JFact is a Java-port of FaCT++, and was chosen mainly to analyse the similarities and differences with its C++ counterpart. While the algorithms are strictly the same, JFact usually is a couple of months behind FaCT++ in terms of up-to-dateness, and therefore not always equal in results. Because it is easier to integrate data types in Java, JFact does support more of them, and may therefore process certain ontologies that FaCT++ outright rejects.

4.8 Supporting Materials and Datasets

All scripts, datasets and some additional materials can be downloaded from the following location:

Material 4.1. *http://owl.cs.manchester.ac.uk/publications/supporting-material/phd-matentzoglu/*

Table 4.1: The methods in which the various stages and tests are recorded. For an explanation of the labelling, see Section 4.1

	JFact
PP	OUTSIDE
CC	TBox.isConsistent();
PRP	TBox.isConsistent()
ST	TBox.createTaxonomy()
POP	TBox.createTaxonomy()
POPFIN	OUTSIDE
SAT	TBox.isSubsHold(Concept,Concept)

	HermiT
PP	OUTSIDE
CC	Reasoner.isConsistent()
PRP	Reasoner.isConsistent()
ST	DeterministicClassification.classify(); QuasiOrderClassification.buildHierarchy();
POP	DeterministicClassification.classify(); QuasiOrderClassification.buildHierarchy();
POPFIN	OUTSIDE
SAT	Tableau.isSatisfiable(boolean, boolean, Set, Set, Set, Set, Map, Map, ReasoningTaskDescription)

	Pellet
PP	OUTSIDE
CC	KnowledgeBase.consistency()
PRP	KnowledgeBase.consistency()
ST	KnowledgeBase.classify()
POP	KnowlegeBase.classify()
POPFIN	OUTSIDE
SAT	ABox.isSatisfiable(ATermAppl, ATermAppl);

Chapter 5

Module Hardness

The core underlying assumption of utilising modules for classification is the *benign module conjecture*[1], which states that no module should be harder to reason with than its parent ontology (see Section 2.4). More formally the conjecture is as follows:

Hypothesis 5.1. *Given an ontology $\mathcal{O}$ and a reasoner $\mathcal{R}$, there is no module $\mathcal{M}$ of $\mathcal{O}$ with $\mathcal{M} \subseteq \mathcal{O}$ such that $CT(\mathcal{O}, \mathcal{R}) < CT(\mathcal{M}, \mathcal{R})$.*

Note that we explore this hypothesis with respect to $\bot$-locality based modules, but the arguments provided for the conjecture in this chapter should hold for all depleting, justification-preserving modules (for example other kinds of locality-based modules). The conjecture is entirely independent of the (sound and complete for $\mathcal{O}$) reasoner and independent of the hardware the classification is performed on—as long as it is fixed (the same reasoner/hardware was used to to obtain the classification time measurement for the module and the whole ontology).

It is trivially the case that there is a module $\mathcal{M}$ such that $CT(\mathcal{O}, \mathcal{R}) \approx CT(\mathcal{M}, \mathcal{R})$ since $\mathcal{O}$ is a module for itself. There are two observations that make this conjecture worth stating.

- It is known that a subset of an ontology can be harder than the ontology itself [GPS12]. This result will also be reconfirmed in this chapter. A module is a subset of an ontology.
- The risk emanating from this possibility is critical. Should the conjecture be violated regularly, a lot of the argument for modular classification would fall apart, or at least become more tricky.

The main goal of this chapter is to show the counter-intuitive result that the conjecture does not hold. As a consequence of that, we will state and test a modified conjecture to establish the actual risk of pathological modules for modular classification. We find that (1) 5.3% of the ontology-reasoner pairs actually

[1] Also internally referred to as the *Bijan conjecture*, after Bijan Parsia.

exhibit pathological modules in a realistic setting, (2) 0.46% of the modules as extracted by state-of-the-are modular reasoning techniques (MORe-modules and Chainsaw maximal modules) exhibit pathological behaviour and (3) the majority of pathological modules correspond to more than 75% of the whole ontology. Our main conclusion is that pathological modules exist and constitute a relatively small threat to modular reasoning techniques that involve *large* modules.

5.1 Definitions and Models

The phenomenon under investigation is the **hardness of the classification of a module**. As extensively discussed in Section 2.4, we focus our attention on syntactic locality-based $\bot$-modules. The **hardness of classification** is the time it takes for a reasoner to classify the ontology, denoted $CT(\mathcal{O}, \mathcal{R})$, or the module, denoted $CT(\mathcal{M}, \mathcal{R})$, operationalised as wall-clock time (Section 4.4.2). Note that the classification time cannot be directly obtained. Instead, we rely on sequences of measurements (runs) on a particular (type of) machine and then choose an appropriate way to obtain a representative value (usually mean, maximum value or median, sometimes first in sequence) based on the sequence that we believe represents the classification time in our context the best. In what follows, we refer to a reasoner as $\mathcal{R}$ and to an ontology as $\mathcal{O}$.

The existence, for example, of Hot Spots [GPS12] suggests that chunking the ontology into smaller bits might have considerable beneficial effects (Section 3.3.3). It was also empirically observed in [GPS12] and has been known anecdotally for years that subsets of an ontology can be harder than the whole. A subset that is harder than the whole is defined as follows:

Definition 5.1. *We say $\mathcal{S} \subset \mathcal{O}$ is a* hard subset *of $\mathcal{O}$ for $\mathcal{R}$ if $CT(\mathcal{O}, \mathcal{R}) < CT(\mathcal{S}, \mathcal{R})$.*

If $\mathcal{O}$ and $\mathcal{R}$ are clear from the context, we simply refer to the subset as $\mathcal{S}$.

Obviously, given an ontology $\mathcal{O}$, a reasoner $\mathcal{R}$, a decomposition $D_{\mathcal{O}}$ (see Section 2.5 and a module $\mathcal{M} \in D_{\mathcal{O}}$, if $CT(\mathcal{M}, \mathcal{R})$ is larger than $CT(\mathcal{O}, \mathcal{R})$, then modular reasoning is counter-productive, and efforts to tune aspects of modular classification are pointless. The goal of this chapter is to investigate the general case: Are modules of an ontology ever harder than $\mathcal{O}$ itself? If so, how prevalent is the phenomenon? In contrast with random subsets, there are some strong arguments for thinking the answer is negative.

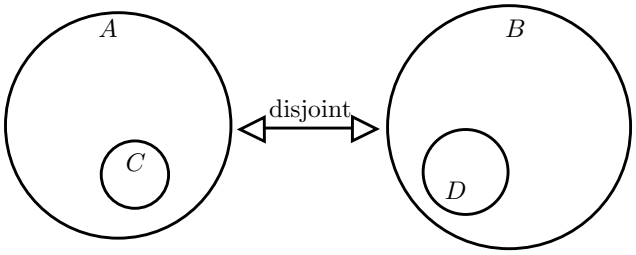


Figure 5.1: Illustration for the effect of a disjointness on subclasses. If A and B are disjoint, than so are C and D.

Firstly, if the reasoner is using standard Enhanced Traversal to classify the ontology, a random subset $\mathcal{S}$ might omit an explicitly asserted subsumption or, perhaps more importantly, a disjointness axiom which considerably prunes the traversal space for atomic subsumptions. Example 5.1 illustrates this phenomenon.

Example 5.1. *Traversal space pruning through disjointness*

- *Given concept names* $A, B, C, D \in \widetilde{\mathcal{O}}$ *with* A *being disjoint from* B, $\mathcal{O} \models C \sqsubseteq A$ *and* $\mathcal{O} \models D \sqsubseteq B$
- *then no* C *can be a subclass of any* D *and vice versa, see Figure 5.1.*
- *The number of tests that can potentially be omitted because of the disjointness is* $2 * n * m$*, where* n *is the number of subclasses of* A *and* m *is the number of subclasses* B*.*

If a disjointness follows from $\mathcal{O}$ but not from $\mathcal{S}$, a reasoner might do a large number of additional tests when classifying the subset, especially if the the disjointness is implied high up in the class hierarchy. Modules must contain all such axioms over their signature in order to capture those entailments.

Secondly, a random subset might break some justification of an atomic subsumption over its signature. For example, some axioms of an "easy" justification for $A \sqsubseteq B$ in $\mathcal{O}$ might be missing in the random subset, leaving only harder justifications unbroken.[2] Examples for such axioms would be the self-justification $A \sqsubseteq B$ (if $A \sqsubseteq B$ was asserted in $\mathcal{O}$), or $\neg A \sqsubseteq \neg B$. A module must contain all justifications for each entailed subsumption.

[2]Harder in the sense that they might for example force the reasoner to construct larger models.

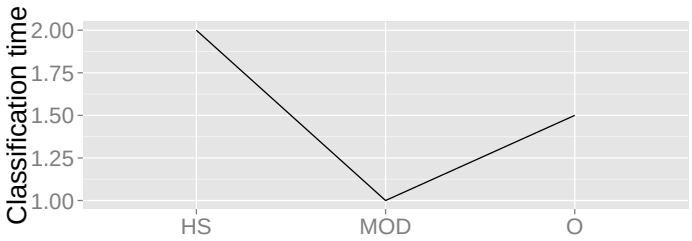


Figure 5.2: Illustration for the expected behaviour of a hard subsets $\mathcal{HS}$ (hs), their corresponding module $\mathcal{M}$ (mod) and the source ontology $\mathcal{O}$ (o). Given that $\mathcal{HS} \subset \mathcal{M} \subset \mathcal{O}$ and $CT(\mathcal{HS}, \mathcal{R}) > CT(\mathcal{O}, \mathcal{R})$ we expect that $CT(\mathcal{O}, \mathcal{R}) \geq CT(\mathcal{M}, \mathcal{R})$.

In general, since a module contains *everything* from the ontology relevant to entailments over its signature (*depletingness*, see Section 2.4), it seems reasonable to think that reasoning should be no harder. No shortcuts can be missing and we have removed a lot of potential distractions. However, these considerations are merely speculative.

If, against our expectation, a module turns out to be harder than the whole ontology, we call it *pathological*, defined as follows:

Definition 5.2. *Given $\mathcal{O}$, $\mathcal{R}$ and a module $\mathcal{M}$ of $\mathcal{O}$ with $\mathcal{M} \subset \mathcal{O}$, we call $\mathcal{M}$ a* pathological module *of $\mathcal{O}$ for $\mathcal{R}$ if $CT(\mathcal{O}, \mathcal{R}) < CT(\mathcal{M}, \mathcal{R})$.*

Figure 5.2 illustrates our expectations for classification performance (based on the argument presented in Section 3.4) with respect to a module $\mathcal{M}$ of the signature of a hard subset $\mathcal{HS}$ and an ontology $\mathcal{O}$ with $\mathcal{HS} \subset \mathcal{M} \subset \mathcal{O}$. If the classification time of the subset $\mathcal{S}$ is larger than the classification time of the whole ontology (hard subset), we expect the classification time of $\mathcal{M}_{\widetilde{\mathcal{S}}}$ to be less than the classification time of $\mathcal{O}$. In other words, modularity can *easify* a formerly hard subset of the ontology by adding axioms. This *protective effect* is defined in the following:

Definition 5.3. *Given $\mathcal{O}$, $\mathcal{R}$, a module $\mathcal{M}$ of $\mathcal{O}$ and a hard subset $\mathcal{HS}$ with $\mathcal{HS} \subset \mathcal{M} \subset \mathcal{O}$, we say modularity has a* protective effect *over $\mathcal{HS}$ if $CT(\mathcal{HS}, \mathcal{R}) > CT(\mathcal{M}, \mathcal{R})$.*

In the analysis, we distinguish between *significantly* and *insignificantly pathological modules*, as defined in the following:

Definition 5.4. *Given $\mathcal{O}$, $\mathcal{R}$ and a pathological module $\mathcal{M}$ of $\mathcal{O}$ with $\mathcal{M} \subset \mathcal{O}$, we say $\mathcal{M}$ is* significantly pathological *if $\frac{CT(\mathcal{M},\mathcal{R})}{CT(\mathcal{O},\mathcal{R})} - 1 \geq 0.05$. A module is insignificantly pathological if it is pathological, but not significantly pathological.*

There are two less obvious, but credible, *arguments against the conjecture*. Firstly, modularity might not play well with other optimisations employed by standard reasoning systems. For example the lower expressivity of $\mathcal{M}$ might trigger a classification procedure implemented on the basis of a different calculus (for example a nested consequence-based procedure), that for some reason is *less efficient* on the module than the primary model-construction procedure. However, this should happen only in extreme cases that are unlikely to occur in practice (for example involving very large modules), we (as reasoner developers) might be able to isolate the dysfunctional interaction with the other optimisations and fix it (or choose to ignore it). Secondly, we know (and reconfirm in Chapter 6) that non-determinism can have a strong effect on classification hardness. It is in principle possible that a module might encourage a reasoner to make worse choices along the way, overall making $CT(\mathcal{M}, \mathcal{R})$ harder than $CT(\mathcal{O}, \mathcal{R})$.

It should be noted that we do not aim to establish the risk of the overall hardness of *all modules in a decomposition*.[3] For example, there are cases of modules that are not pathological, but are not easy enough to justify the employment of modularity, even if we disregard the overhead induced by computing the decomposition. If the classification time of the module is just a bit short of the classification time of the ontology, any gains from modularity are marginal, and most likely outweighed by the effort of classifying the module(s) representing the remaining ontology (even if they do not overlap).

As a reminder, an OWL 2 DL ontology is one that is *not* OWL Full, a *profiled ontology* is one that falls under one of the three OWL 2 profiles (QL, EL, RL) and a *pure DL ontology* is one that is neither OWL Full, nor profiled.

5.2 Empirical Characterisation

The primary goal of this chapter is testing Conjecture 5.1 for modules that are *at least potentially non-trivial in terms of total classification time*.[4] This systematic bias was introduced because of the high degree of measurement variance in the

[3] For details on what a decomposition is, see Section 2.5.

[4] Operationalised in Section 5.3.1.

sub-second area and tight resource constraints. This bias poses a threat to the external validity of the results: It is possible that pathological cases only occur in modules that can be classified in less than 10 seconds. Since we do not claim to measure the density of pathological modules in general, but merely to verify their existence, this bias is acceptable.

A full formal understanding of OWL ontology hardness for a reasoner is not in sight (see Section 1.3), but we can gain some sound understanding by resorting to empirical methods. The full empirical characterisation of module hardness consists of two main parts:

1. Targeted search for pathological modules
2. Testing the *benign module conjecture*

We will illustrate the problem of taking a random sample of modules and propose a solution to finding potentially hard ones.

5.2.1 Method: Finding Pathological Modules

We can only be sure that Conjecture 5.1 holds if we classify all modules $\mathcal{M} \subset \mathcal{O}$ with all possible reasoners $\mathcal{R}$ and do not find a single *pathological module*, see Definition 5.2 (assuming no experimental error). To clearly reject the conjecture we restrict ourselves to significantly pathological modules, as measurement variance might make differences of less than 5% questionable, especially if the module is almost as big as the ontology. As we will see in Section 5.6.1, the variance across runs exceeds 5% only in a few (1.54%) cases. Therefore we consider the 5% threshold as safe enough to determine a "truly" pathological module.

The *main problem* here is that the set of all $\mathcal{M} \subseteq \mathcal{O}$ for any given $\mathcal{O}$ is potentially exponential in the number of axioms in $\mathcal{O}$ [DVPSS11b]. This theoretical threat has been shown to hold in practice [DVPSS11b, PS10]. The authors of [DVPSS11b] describe in detail pessimal and optimal patterns of ontologies leading to from between 1 up to exponentially many total modules in $\mathcal{O}$ and suggest that out of their corpus, most ontologies exhibit exponential behaviour. In fact, only two of their ontologies (both with fewer than 50 axioms and 25 names) could be fully modularised, i.e. *all* modules extracted, within a reasonable timeout of "several hours". Even if we were able to extract all modules for some $\mathcal{O}$, we might not be able to classify them all due to time constraints. Consider an ontology $\mathcal{O}$ with a signature $\widetilde{\mathcal{O}}$ containing more than 1,000 names and at least 1,000 axioms. The number of modules for this ontology exceed, in the worst case, $2^{1000} - 1$

(number of k-combinations for all k) or, in other words, a number with 302 digits (the worst case is the case where no two axioms have overlapping signature).[5] Even if we assumed that there are as many as 2^{10} pathological modules in $\mathcal{O}$ (an arbitrary guess to illustrate the problem), the probability for a random module to be pathological is tiny ($9.557 * 10^{-299}$), and thus finding even one of them through random sampling will be unlikely. While the worst case is perhaps not a realistic measure, we know from previous work around logical decompositions (see Section 2.5) that some ontologies are comprised of a large number of logically disconnected components [DVPSS11b], a strong indicator for a large number of potential modules. Moreover, the example understates the size of many ontologies by one to four orders of magnitude, both in terms of signature and axioms.

A solution to this problem might be to take a *representative* sample of all the modules in $\mathcal{O}$. The only realistic way to obtain a random sample of modules is to compute random sub-signatures $\mathcal{S} \subset \widetilde{\mathcal{O}}$ to be used as seeds to extract the module. The normal way to obtain a random $\mathcal{S}$ would be to iterate through the signature, and for each name, flip a coin. For heads, the name is included in $\mathcal{S}$, for tails, not. It can easily be seen that a non-stratified random sample would almost exclusively contain sub-signatures of size $\frac{|\widetilde{\mathcal{O}}|}{2}$. This is problematic because we know from experiments with random subsets [GPS12] that hard ones are generally quite large, more often than not around $\frac{7}{8}$ of $|\mathcal{O}|$. A purely random sample would not guarantee us to find any modules in that size range, a systematic bias that might damage the internal validity of the study.

Another potential sampling approach might be *stratified random sampling*. However, a random sample stratified by the size of the signature (a random sample of signature size 1, 2,..k..,n) would still be huge. Even if we deem 1 single random signature of size k as sufficiently representative of all signatures of size k (which it is probably not), we would still have $|\widetilde{\mathcal{O}}|$ modules to process. In our optimistic example, we would have to process 1000 modules (for one ontology), each of which is potentially as hard as the whole ontology. Classifying so many modules is *neither practical, nor likely to be profitable*, let alone executing each multiple times to mitigate experimental error. An alternative to simple stratified sampling might be an approach involving a *weighted coin*. This way, the size of the seed signature could be biased towards larger numbers, therefore increasing

[5] Even if every module would take only a millisecond to classify, it would take longer to classify all modules than the universe has existed so far.

the likelihood of larger and potentially harder modules. However, we do not know whether larger modules are generally harder than smaller ones (we only assume that), which would make any attempts to bias the coin more or less arbitrary. Even if we would obtain a statistically significant (stratified) sample, the (very likely) low probability of encountering pathological modules would not provide us with any certainty with respect to their existence.

In order to increase the chance of finding pathological modules, we employ the following *heuristic*. We know that reasoners are *performance heterogeneous* over a wide range of ontologies [GPS12], we know that hard subsets exist [GPS12], and we do know how to find them. From every ontology, a number of random *paths*, Definition 5.5, are sampled to identify proportions of ontologies which might exhibit *hard subsets*.

Definition 5.5. *Given $\mathcal{O}$ and a sequence of n subsets $\mathcal{S}_i \subseteq \mathcal{O}$ we call $(\mathcal{S}_i)_{i=1}^n$ an* n-path *of $\mathcal{O}$ if*

- $\mathcal{S}_1 \subset \mathcal{S}_2 \subset ... \subset \mathcal{S}_{n-1} \subset \mathcal{O} = \mathcal{S}_n$ *and*
- $|\mathcal{S}_i| = \frac{i}{n} * |\mathcal{O}|$.

A path is sampled by *slicing* the ontology $\mathcal{O}$ into subsets of size $\frac{1}{n}$, in our case with $n = 8$. I.e, the first slice corresponds to $\frac{1}{8}$ of the ontology, the first two slices together to $\frac{2}{8}$ and so on, the full ontology to $\frac{8}{8}$. Each $\frac{m}{8}$ subset is called a *slice* of the ontology, the index m representing a *proportion* for the ontology, defined in the following:

Definition 5.6. *Given $\mathcal{O}$ and two counting numbers m and n with $0 < m \leq n$, the* $\frac{m}{n}$-proportion *of $\mathcal{O}$, written $\mathcal{O}_n^m$, is defined as follows:*

$$\mathcal{O}_n^m := \{\mathcal{S} \subseteq \mathcal{O} \mid |\mathcal{S}| = \frac{m}{n} * |\mathcal{O}|\}$$

In other words, $\mathcal{O}_n^m$ represents the set of all subsets of $\mathcal{O}$ of size $\frac{m}{n}$. For an example of a classified path, see Figure 5.4 later in the chapter. A path $(\mathcal{S}_i)_{i=1}^n$ is called *monotonic* if $CT(\mathcal{S}_i, \mathcal{R}) < CT(\mathcal{S}_{i+1}, \mathcal{R})$ for each $i < n$. In other words, a monotonic path raises only upwards towards $CT(\mathcal{O}, \mathcal{R})$. Other metrics related to performance heterogeneity, can be found elsewhere [GPS12].

For every hard subset $\mathcal{HS}_i$ we extract $\mathcal{M}_i$, defined as $\mathcal{M}_i := \bot\text{-mod}(\widetilde{\mathcal{HS}}_i, \mathcal{O})$. It follows that $\mathcal{HS} \subseteq \mathcal{M} \subseteq \mathcal{O}$, assuming that $\mathcal{HS}_i$ has no tautologies. If the module hardness conjecture holds, for any $\mathcal{HS}, \mathcal{M}, \mathcal{O}$ such that $\mathcal{HS} \subseteq \mathcal{M} \subseteq \mathcal{O}$

and any reasoner $\mathcal{R}$ it should follow that $CT(\mathcal{HS}, \mathcal{R}) \geq CT(\mathcal{O}, \mathcal{R}) \geq CT(\mathcal{M}, \mathcal{R})$. A case for which $CT(\mathcal{M}, \mathcal{R}) > CT(\mathcal{O}, \mathcal{R})$ is called a *pathological case*, or a case refuting the conjecture. In other words, we are trying to find evidence against the conjecture by taking subsets of the ontology that are harder than the whole, add 0 or more axioms to them to turn the subset into a module and then check whether this process dropped the hardness of the module to below (or equal to) the hardness of the ontology.

We want to emphasise at this point that this targeted search strategy is *not the only possible approach* to bias towards (potential) pathological modules. Another option would be to bias random module sampling towards ontologies involving hard reasoning problems (hard subsumption tests, Hot Spots) or a high degree of observable non-determinism, assuming that only those have the potential to actually lure the reasoner into a harder space. There is no conclusive reason to pick one strategy over the other, but we decided that the potential of observing a *protective effect* of modularity using the hard subset method was by itself appealing enough to pick the hard subset based method.

5.3 Experimental Design

In the following we will discuss Experiment 5.1.

Experiment 5.1. *Testing the benign module conjecture by extracting modules using hard subsets.*

For this experiment, we use Pellet, FaCT++ and HermiT, see Section 4.7.2. JFact had to be excluded because of resource limitations. Its role as a Java-port of FaCT++ made it the obvious first choice for exclusion. We conducted our study on the BioPortal snapshot described in Section 4.6.3 and performed the classifications on our Mac Mini cluster (Machine 4.1). Note that we perform the experiments on all of BioPortal, including OWL Full ontologies. Standard DL-reasoners are incomplete over OWL Full ontologies in a *not externally specified way*, which makes their behaviour mostly incomparable. We clearly isolate both DL from non-DL cases in the analysis.

5.3.1 Experimental Pipeline

The method introduced in Section 5.2.1 has six steps:

Procedure 5.1. *Finding pathological modules*

1. *Extract* paths *from* $\mathcal{O}$
2. *Identify* hard proportions *of* $\mathcal{O}$
3. *Sample randomly from* hard proportions *in* $\mathcal{O}$
4. *Identify* hard subsets
5. *Extract modules from hard subsets signature*
6. *Identify* pathological modules

In the following, we will provide some details of our pathological module finding method (Procedure 5.1). For every ontology in the corpus we extracted *three different paths* (Step 1).[6] We conducted the investigation on all BioPortal ontologies with more than 10 axioms (328). We started by drawing a random eighth of the set of (logical) axioms in $\mathcal{O}$ and exported it as a new ontology. Then a second eighth of the remaining axioms was drawn (randomly) and added to the first eighth to get the $\frac{2}{8}$ slice of the ontology, which is again exported as a new ontology. This procedure is repeated for $\frac{3}{8}$, $\frac{4}{8}$ and so on until the last random subset of $\mathcal{O}$, $\frac{7}{8}$. A *path* thus consisted of eight cumulatively grown subsets, the $\frac{8}{8}$ corresponding to the full ontology. We then classified all 6888 subsets (7 subsets per path, 3 paths per ontology, 328 ontologies) and the 328 full ontologies with all three reasoners. We call the first classification time obtained for a full ontology the *base case* as defined in Definition 5.7.

Definition 5.7. *Given* $\mathcal{R}$*,* $\mathcal{O}$ *and a sequence* $CT(\mathcal{O}, \mathcal{R}, i)$ *of classification time measurements for* $\mathcal{O}$ *by* $\mathcal{R}$ *we call* $CT(\mathcal{O}, \mathcal{R}, 1)$ *the* base case*.*

Next, we determine the set of *hard proportions* in $\mathcal{O}$ (Definition 5.8) by looking for respective *witnesses*.

Definition 5.8. *Given* $\mathcal{R}$ *and* $\mathcal{O}$ *we call* $\mathcal{O}_n^m$ *a* hard proportion *of* $\mathcal{O}$ *with respect to* $\mathcal{R}$ *if there is at least one witness for the hard proportion with respect to* $\mathcal{O}$ *and* $\mathcal{R}$*.*

We call $\mathcal{S} \subseteq \mathcal{O}$ *a* witness *for the* $\frac{|\mathcal{S}|}{|\mathcal{O}|}$*- proportion of* $\mathcal{O}$ *if*

1. $CT(\mathcal{S}_i, \mathcal{R}) > CT(\mathcal{O}, \mathcal{R})$
2. $CT(\mathcal{S}_i, \mathcal{R}) > 10$ *seconds*

Condition 2 for being a witness (Definition 5.8) was employed to mitigate resource limitations and to ensure that classification time differences between

[6]The number three is often, as in this case, the sweet spot between feasibility and experimental error mitigation.

modules and subsets could be high enough to be attributed to actual phenomena (rather than measurement error). We then sampled 20 random subsets of size $\frac{i}{8}$ of $|\mathcal{O}|$ for every hard proportion of $\mathcal{O}$ (Step 3). For example, if we found during the path screening that a subset $\mathcal{S}_5 \subset \mathcal{O}$ was harder than $\mathcal{O}$, we would have sampled 20 random subsets of size $\frac{5}{8}$ of $|\mathcal{O}|$. We then classified these subsets with the reasoner that witnessed the hardness. For example, if HermiT had a hard subset $\mathcal{S}_5 \subset \mathcal{O}$, we had HermiT (and only HermiT) classifying the 20 random subsets of size $\frac{5}{8}$ of $|\mathcal{O}|$. Restricting the classification of a subset to the reasoner that witnessed the hardness was necessary because of resource limitations, and the large differences in terms of performance heterogeneity between the reasoners [GPS12]. To mitigate the effect of experimental error, we considered as hard subsets only those that proved harder than their source ontology in three independent experiment runs. To save further computational resources, we classified all sampled subsets once; if they appeared *hard*, a second time, and if hard again, a third time. Of all hard subsets found this way, we extracted their signature and used them as a seed to extract the corresponding $\bot$-module using the OWL API (Step 5). We then classified the resulting modules three times each. If, for a module $\mathcal{M}$ of $\mathcal{O}$, we found that $CT(\mathcal{M}, \mathcal{R}) > CT(\mathcal{O}, \mathcal{R})$ in three independent runs, we classified $\mathcal{M}$ as a *pathological module* (Step 6).

5.4 Results

Out of the 330 ontologies in the corpus (see Section 4.6.3), paths were extracted for the 328 with more than 10 logical axioms. In total, 6,888 subsets were extracted (328 ontologies * 7 subsets per path * 3 paths per ontology, full ontologies are treated separately in the following), which led to 20,664 attempted classifications (6,888 subsets * 3 reasoners). Out of these, 18,978 were successfully completed (92%) within a timeout of 30 minutes. HermiT completed 6,374 (93%), Pellet 6,399 (93%) and FaCT++ 6,205 (90%). Out of the 1,686 failed classification attempts, more than 46% were due to timeout, 12% due to unsupported datatypes, 5% due to inconsistency and the rest (mostly) reasoner internal failures. Out of the 328 *full ontologies* and 984 classification attempts (3 reasoners * 328 ontologies), 834 (84%) successfully concluded (HermiT and Pellet 283 (86%) and FaCT++ 268 (82%)).

5.4.1 Finding Pathological Modules

Out of the 984 paths per reasoner (328 ontologies, 3 paths each), 804 paths were fully completed for FaCT++, 845 for HermiT and 829 for Pellet. We excluded from our analysis all paths for which we could not obtain the base case, see Section 5.3.1. 145 paths for FaCT++, 124 for Pellet and 110 paths for HermiT were excluded because we did not obtain a base case ($CT(\mathcal{O})$), reducing the 18,978 successful classifications of hard subsets by 1533 to a total 17,455.

42% of all paths completed by FaCT++ are *non-monotonic*, 24% of paths for HermiT and 60% of paths for Pellet. This suggests that HermiT is somehow more *stable* in terms of reasoning performance than the other two reasoners, potentially because it is less affected by non-determinism [GHM+14]. Understanding the differences between reasoners will be part of future work.

Identifying Potentially Hard Proportions of Ontologies

Of the 17,455 potential *witnesses* for hard proportions of $\mathcal{O}$ (see Section 5.3.1), 633 involved a subset that had a classification time of more than 10 seconds (see Definition 5.8, condition 2). Figure 5.3 shows a complete break-down by language family and reasoners. Of these 633 potential witnesses, 143 (22.6%) were harder compared to their base case (Definition 5.8, condition 1). In total, these 143 cases were witnesses to 52 unique potentially hard proportions of $\mathcal{O}$, including 34 for FaCT++, 16 for Pellet and 2 for HermiT. The remaining 490 cases involving subsets harder than ten seconds were witnesses to 185 unique non-hard proportions, including 36 for FaCT++, 52 for Pellet and 97 for HermiT. Figure 5.4 (see later in this chapter) shows the paths for the 5 DL ontologies that turned out to contain hard modules, broken down by reasoner.

Hard subsets could be found at almost all proportions of $\mathcal{O}$. Restricting ourselves to DL parent ontologies, we can see in Table 5.1 that unsurprisingly, the probability of a subset being hard grows the larger the proportion. It is interesting that the relative number of *hard subsets* across all proportions of $\mathcal{O}$ differs only a bit between subsets with $CT(\mathcal{S}, \mathcal{R}) > 10$ sec and those $CT(\mathcal{S}, \mathcal{R}) \leq 10$. This suggests that the bias introduced by considering only subsets harder than 10 seconds does not hurt generalisability too much.[7]

[7] Which is nice to know, but not strictly necessary as we do not claim generalisability.

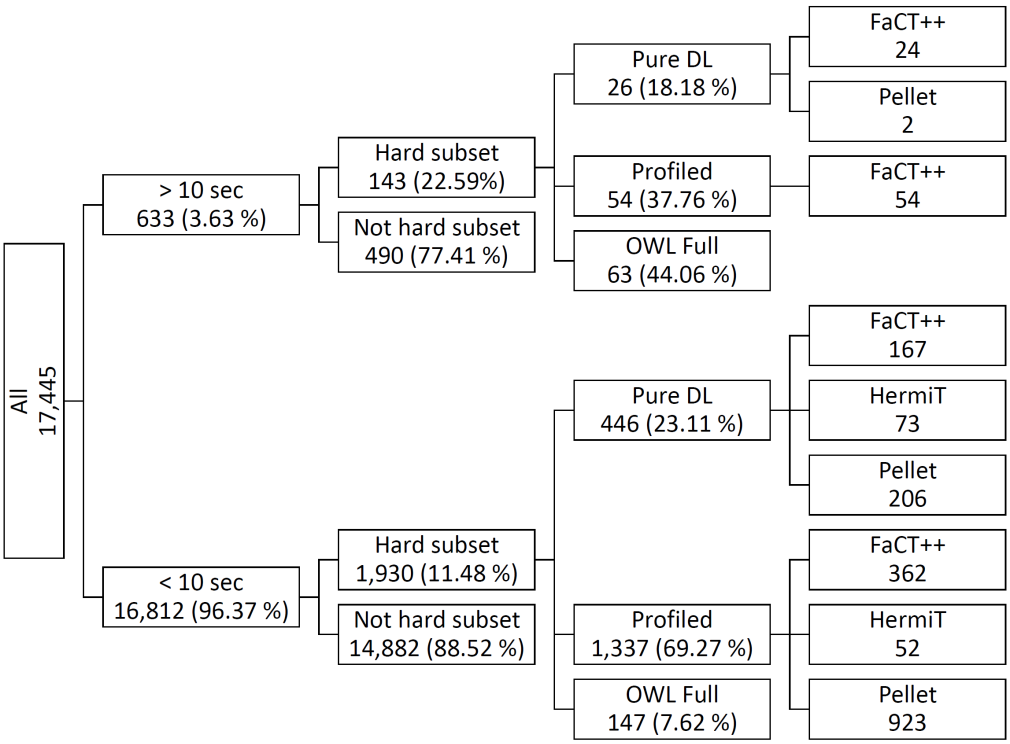


Figure 5.3: Break-down of random subsets obtained through random path sampling. For example, we can observe that out of the 17,445 subsets classified in total, 633 were harder than 10 seconds. Out of these 633, 143 were actually hard.

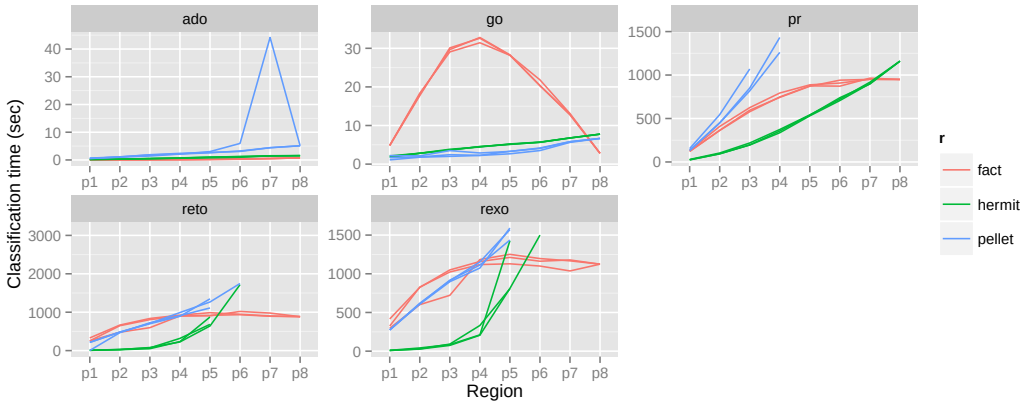


Figure 5.4: Paths for OWL 2 DL ontologies with hard modules. Each line represents a path. On each path, we have a measurement point coming from a single classification of the random subset of the respective proportion of $\mathcal{O}$. For example, the peak in the first plot (ADO ontology) came from a classification of a random subset of size $\frac{7}{8} * |\mathcal{O}|$ by Pellet.

Prop.	#All	#Hard	#Easy	%All	%Hard	%Easy
1	72	0	72	4%	0%	4%
2	115	9	106	6%	11%	6%
3	149	9	140	8%	11%	8%
4	213	14	199	11%	18%	11%
5	278	15	263	15%	19%	15%
6	397	15	382	21%	19%	21%
7	639	18	621	34%	22%	35%
All	1863	80	1,783	100%	100%	100%

Table 5.1: Number of hard subsets by proportion of $\mathcal{O}$, broken down by easy ($CT(\mathcal{S}, \mathcal{R}) \leq 10$ sec) and hard ($CT(\mathcal{S}, \mathcal{R}) > 10$ sec).

Determining Hard Subsets

Next, we describe the results of the classification of the 20 randomly sampled subsets for each hard proportion of $\mathcal{O}$. Out of the 1040 subset classification attempts (52 hard proportions, 20 random subsets each), 921 (88.6%) turned out to be hard after the first run, 903 (86.8%) after the second and 900 (86.5%) after the third. HermiT only ever witnessed a hard subset for OWL Full ontologies, Pellet and FaCT++ witnessed hard subsets both for Full and DL ontologies. Out of the 52 potentially hard *proportions* of $\mathcal{O}$ we started with, 49 turned out to be actually hard (i.e., containing hard subsets). Out of these, for 37 proportions *all* measured subsets turned out to be hard, and only 3 subsets had less than 5 witnesses. Out of the 900 cases of hard subsets, 401 came from OWL Full ontologies (44.6%), 360 from OWL 2 DL ontologies falling under one of the three profiles (40%) and only 139 (15.4%) from pure OWL 2 ontologies. All 900 cases are more or less evenly distributed across all proportions (1-7) across 10 different ontologies.[8] Expressivity levels of these 10 ontologies range from $\mathcal{ALE}$ to $\mathcal{SHOIQ}(\mathcal{D})$. One observation to note is that most of the source ontologies are large: the smallest (ADO) has 2,401 axioms, 2 ontologies are around 33,000 and 80,000, and the remaining 7 above 100,000 axioms. 2 ontologies (EP and GLYCO) have large proportions of ABox axioms. The size of the ontologies might be a consequence of the 10-second filter introduced for the hard subset finding, i.e. subsets that take longer than 10 seconds to classify should usually be of a considerable size themselves.

[8] ADO, CHEBI, EP, GLYCO, GO, HINO, PR, RETO, REXO, RH-MESH

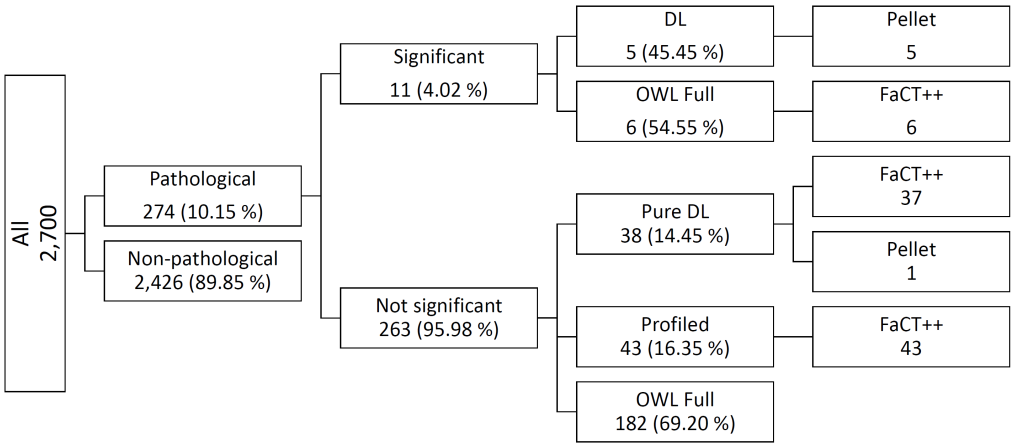


Figure 5.5: Breakdown of pathological cases for modules sampled from hard subsets.

Testing for Hard Modules

After extracting $\bot$-modules for the 900 subset-reasoner pairs and running them 3 times each (2700 classifications), we ended up with 274 classifications (10.2%) resulting in pathological modules when compared to the base case of their respective parent ontology. If we exclude OWL Full ontologies (1494 remaining), the share of hard module cases drops to 5.8% (among OWL 2 DL ontologies, 86 cases). Out of these, only 5 (0.34%) are significantly pathological. The protective effect of modularity can be said to be 99.66%, or in other words, 99.66% of the hard subsets from OWL DL parent ontologies were *protected* by turning them into modules. The majority of these 86 cases are produced by GO and RETO using FaCT++ with 43 and 29 cases, respectively. Hermit did not produce any pathological cases within OWL 2 DL (in fact, not even a hard subset). Pellet contributed 6 pathological cases to the 86 with modules from the ADO ontology, all 6 differing by merely fractions of a second (making up about 95% of the size of $\mathcal{O}$). Only 5 of the 86 cases involve significantly pathological modules, all of which involving modules that constitute around 95% of the size of their parent (all Pellet on ADO), see Figure 5.5. The distribution of pathological modules can be seen in Figure 5.6.

Cross-checking Results

We compared the classification time of a module (every run) to only a single run of the full ontology classification, which may have led to measurement error.

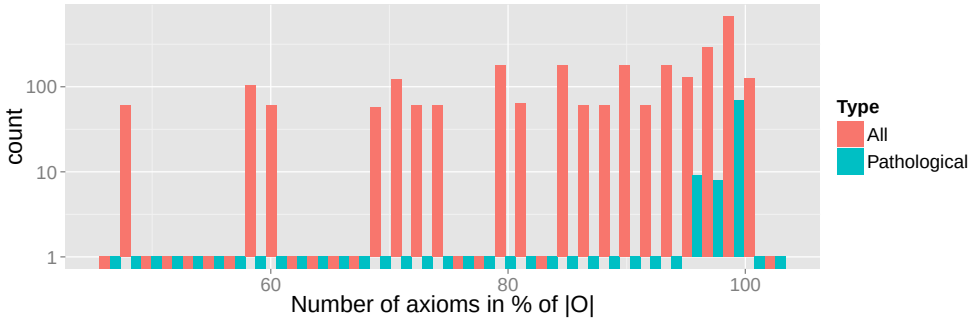


Figure 5.6: Histogram showing the distribution of module sizes compared to their parent ontology, excluding OWL Full (x-axis, in %). Pathological modules include insignificantly pathological. Because of the small number of pathological modules, the y-axis of the histogram is presented in log-scale.

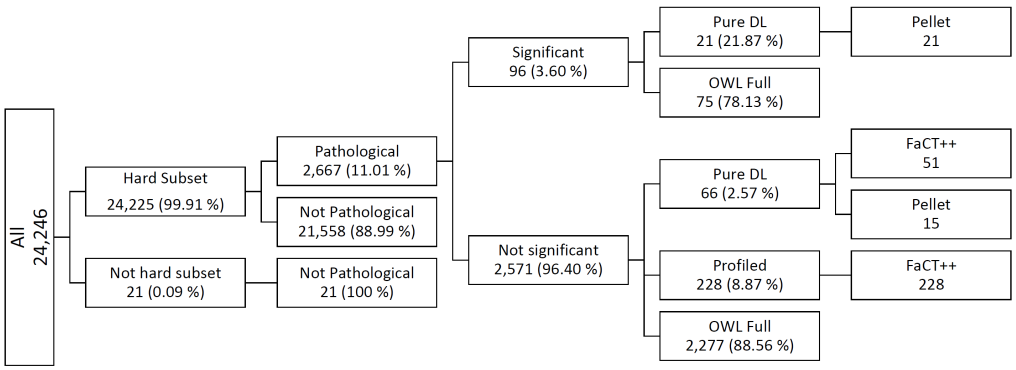


Figure 5.7: Overview of pathological cases for the cross comparison of all modules, hard subsets and parent ontologies.

To corroborate the previous results, we will cross-compare our measurements for hard subsets and modules with three repeated classifications of $\mathcal{O}$ (discarding the base case entirely). With 2700 runs of hard subsets (900 pairs, 3 runs), 3 runs of module classifications, 3 runs of $CT(\mathcal{O})$, that makes 24,300 total comparisons (900 * 3 * 3 * 3). Out of these, we obtained 24,246 full comparisons for the following analysis.

Figure 5.7 shows the breakdown into pathological cases by reasoner and profile. The main thing to take away here is that, again, only Pellet exhibited a number of *significant* pathological cases for OWL 2 DL source ontologies. In fact, the break-down of the cross compared data set appears quite similar to the comparison against a single base case, Figure 5.5. This suggests that at least in our case, a single base case is representative.

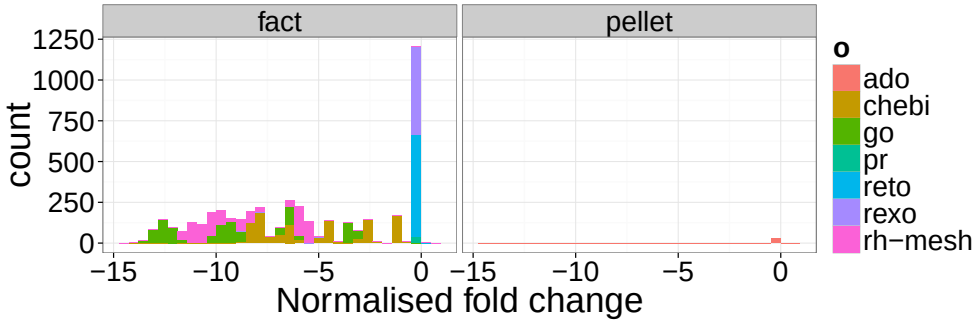


Figure 5.8: Histogram illustrating the protective effect of modularity. The x-axis shows the protective effect, quantified by the relative difference (fold change) in classification time between the hard subset and the module. For example, the RETO and REXO modules were only marginally easier for FaCT++ than their respective hard subsets, while GO modules were up to 14 times easier.

Protective Effect

Figure 5.8 illustrates the protective effect (see Section 5.1) of modules over hard subsets, broken down by ontology (excluding OWL Full). The protective effect can only be, at least with our data, illustrated by looking at FaCT++ because it exhibited by far the majority of hard subsets among DL parent ontologies. While the effect is very small for the majority of the cases (RETO, REXO and PR ontologies), we can see quite a number of significant easyfications for GO, RH-MESH and CHEBI, sometimes more than 14 fold.

5.5 Modified Benign Module Conjecture

Given that we did find 86 pathological and 5 significantly pathological cases (compared to the base case, Figure 5.5) for OWL 2 DL ontologies using our search strategy, we have to reject the *benign module conjecture*. This poses the question of how dangerous this observation is for modular reasoning techniques. In order to address this threat, we investigate the following modified conjecture:

Hypothesis 5.2. *Given an ontology $\mathcal{O}$ and a reasoner $\mathcal{R}$, there is no module $\mathcal{M} \subseteq \mathcal{O}$* extracted by a modular reasoning technique based on the MORe or the Chainsaw approach *such that the classification time of the ontology $CT(\mathcal{O}, \mathcal{R})$ is smaller than the classification time of the module $CT(\mathcal{M}, \mathcal{R})$.*

The main reason why this conjecture is more likely to hold is that all pathological modules found by our targeted search are larger than 90% of their parent ontology, sometimes being just a few axioms short of $\mathcal{O}$ (Figure 5.6). Our experience with modular reasoning suggests that the modules we are dealing with are typically smaller than that. In the following we describe Experiment 5.2.

Experiment 5.2. *Testing the modified benign module conjecture by extracting modules used by state-of-the-art modular reasoners.*

5.5.1 Experimental Pipeline

In order to address our risk for realistic cases, we

- extracted both MORe-modules and sampled randomly from the set of Chainsaw-modules (see Section 3.3.2) across all BioPortal-ontologies
- classified all modules *once*[9] with HermiT, Pellet and FaCT++.
- compared classification times to the *base case*.

We attempted to extract both $\mathcal{EL}$ and remainder modules of the MORe-decomposition, see Section 3.3.1, across our corpus within in a timeout of 12 hours. Because we were only interested in cases where the modules were neither empty nor equal to $\mathcal{O}$, we only considered those pairs of modules for which both modules were non-empty.

In order to extract all *maximal modules* we first decomposed and serialised the Atomic Decomposition of all BioPortal-ontologies within a timeout of 12 hours.[10] We then randomly sampled 30 top atoms per ontology and extracted and serialised the corresponding *maximal modules*. See Section 3.3.2 for how Chainsaw uses maximal modules.

All modules extracted were classified (once) by all three reasoners described in Section 5.3.1, allowing a 30 minute timeout per classification (timeout, as always, due to resource constraints). The order of executions was again fully randomised and distributed across all four machines in the cluster.

5.5.2 Results

From the 24,423 attempted module classifications, 23,418 successfully terminated (96%). For testing Conjecture 5.2, we exclude a further 2,122 classifications, for

[9]Note the potential for experimental error.
[10]See elsewhere [MP14c] for details about the technique

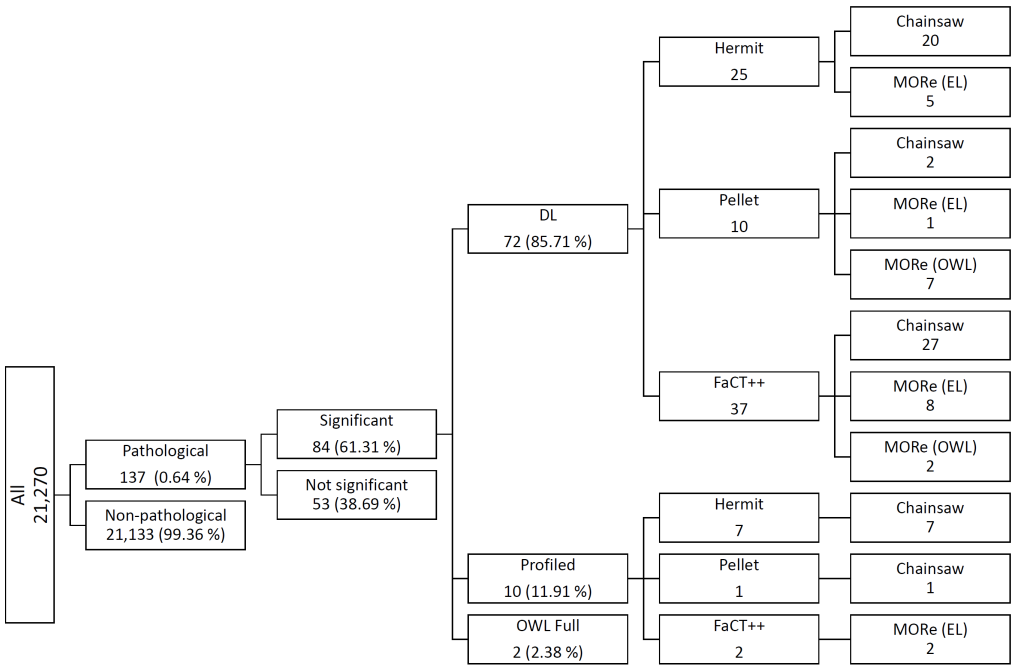


Figure 5.9: Overview of pathological cases for MORe and Chainsaw modules.

which we did not obtain a base case (see Section 5.3.1) and 26 for which the size of the module was equal to the size of the ontology. Out of the remaining 21,270 (2,319 from OWL Full, 11,373 from OWL 2 EL/QL/RL, 7,578 from pure OWL 2 DL parent ontologies), 137 (0.6%) violate the conjecture (11 from OWL Full, 22 from OWL 2 EL/QL/RL, 104 from pure OWL 2 DL ontologies, see Figure 5.9). Out of these, 84 had significant measurement differences (between $\mathcal{O}$ and $\mathcal{M}$) of more than 5%: 2 from OWL Full, 10 from OWL 2 EL/QL/RL, and 72 from pure OWL 2 DL ontologies. 22 out of the 72 pure OWL 2 DL ontologies had differences harder than a second (0 out of the ones falling under one of the profiles). Figure 5.9 shows a detailed breakdown by reasoner and module type. If we ignore OWL Full, **0.43% of the realistic cases are significantly pathological** (82 cases). 36 out of the 681 ontology-reasoner pairs (5.3%) in the sample exhibit pathological behaviour.

Significantly pathological modules are between 1.05 and 34.37 times (median 1.13, 3rd quartile 1.26) harder than their parent ontology. In only 5 cases from DL ontologies was the classification time of the module more than twice as high as that of the parent ontologies (one case of FaCT++ with MHC, 4 of HermiT with PORO).

In terms of parent ontology, only three OWL 2 DL ontologies, namely CAO

(31 cases), NEMO (2 cases) and NPO (2 cases) had more than one reasoner experiencing a significantly pathological module. Interestingly, 2 reasoners experienced a *pathological* MORe OWL remainder module for NEMO and NPO. This is a warning sign for this modular reasoning technique, as MORe-remainder modules are classified by a full fledged OWL 2 reasoner such as the ones in our experiment. Overall 14 out of 191 cases (7.3%) involving MORe-remainder modules from OWL 2 DL parent ontologies in the sample are significantly pathological, 5 out of which have more than a second difference in classification time (2 Pellet, 3 HermiT). 57 out of 18,569 (0.3%) Chainsaw-modules from OWL 2 DL parent ontologies exhibited significantly pathological behaviour, 13 out of which have more than a second difference in classification time (all HermiT, 10 in CAO and 3 in PORO).

Note that differences in classification time can often be attributed to the effects of non-determinism. We will present a more detailed analysis of the non-deterministic behaviour of traversal in Chapter 6.

In terms of size, 28 out of the 82 (34.2%) significantly pathological cases involve modules that constitute more than 90% of (the size of) $\mathcal{O}$. 7 cases (8.54%) involve modules between 75% and 90% of $\mathcal{O}$, 39 (47.56%) between 50% and 75%, 4 (4.88%) between 25% and 50%, 3 (3.66%) between 10% and 25% and only 1 case (1.22%) involves a modules smaller than 10% of $\mathcal{O}$.

One interesting *side-observation* is that there seems to be a slightly higher likelihood of finding a significantly pathological module in ontologies that by themselves take longer than 10 seconds to classify (1.75%) compared to ontologies which take less than 10 seconds to classify (0.37%). The likelihood of encountering an insignificantly pathological module is with around 0.23% the same for both groups.

Figure 5.10 show the difference of pathological modules to real modules as histograms. While the majority of real modules are clearly small, many of the harder modules are more than 60% of the size of their source ontologies.

5.6 Discussion

The viability of modular techniques to optimise OWL classification is based on the assumption that reasoning with a module $\mathcal{M} \subseteq \mathcal{O}$ is never harder than reasoning with the whole ontology. This fundamental assumption was tested thoroughly

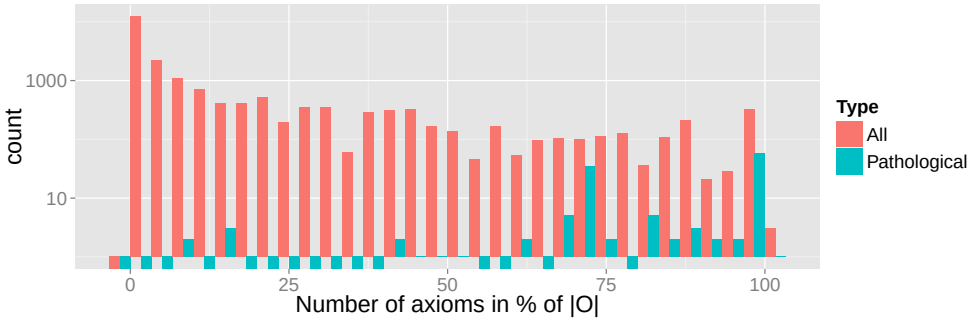


Figure 5.10: Histogram showing the distribution of (Chainsaw/MORe-)module sizes compared to their ontology, excluding OWL Full (x-axis, in %). Pathological modules include insignificantly pathological. Because of the small number of pathological modules, the y-axis of the histogram is presented in log-scale.

in the course of this chapter and was, at least in all generality, rejected. We found however that the significantly pathological modules resulting from our first targeted search strategy (Experiment 5.1) were large, more than 90% of the size of the whole ontology. From prior experience we held the belief that modules encountered by actual modular classification techniques were significantly smaller, which led us to modify our conjecture to generalise only to modules as they are encountered by MORe and Chainsaw. We confirmed that such modules were significantly smaller than the ones we found through our targeted search. Despite that, perhaps surprisingly, we found that such modules are occasionally pathological; this time scattered more widely across the size spectrum. This poses a threat for modern modular reasoning techniques.

As of yet, we do not have a good understanding of why a module is pathological. Our best guess is that *modules that are quite large fractions* of their parent ontology may be pathological merely due to harmful stochastic effects. In Chapter 6 we confirm that classification behaves non-deterministically for a large number of ontologies. Determining whether algorithmic non-determinism *causes* pathological stochastic behaviour can only be tested by actually looking at the models created along the way, and will be part of future work. However, there are some preliminary indications that detrimental stochastic effects and module pathology are related. Significantly pathological modules typically occur outside of the three OWL 2 profiles, with only a handful of exceptions, while hard subsets are also found among profiled ontologies. Algorithmic non-determinism (in our case mostly OR-branching) can be a major cause of harmful stochastic

effects.[11] As higher levels of expressivity are typically associated with higher degrees of non-determinism, it is possible that the the higher frequency in which we find pathological modules outside the polynomial profiles might be explained by the higher potential for stochastic effects in those kinds of modules. The fact that HermiT does not find any pathological modules in the first experiment for example could be explained by the fact that many axioms that would cause non-determinism for the other three reasoners might not do so for HermiT (essentially anything that can be rewritten into Horn), thereby reducing the overall probability for detrimental stochastic effects.

For smaller modules, the explanations get trickier. As we will see in Chapter 6 non-determinism can have occasionally significant consequences, but tests are generally very easy, and variation of classification times between different runs (of the same thing) rarely exceed 5% (see also next section). Therefore, we are *less confident* to suggest that small pathological modules, such as some of the ones we found, are pathological due to non-determinism alone. Another possibility that needs to be evaluated as part of future work is the question of whether the drop in expressivity from $\mathcal{O}$ to $\mathcal{M}$ has forced the reasoner into a "bad" state. For example, given an ontology $\mathcal{O}$ in $\mathcal{SROIQ}$ and a module $\mathcal{M}$ in $\mathcal{EL}$ with $\mathcal{M} \subset \mathcal{O}$, the primary model-construction-based procedure might deal with $\mathcal{O}$ and therefore its subset $\mathcal{M}$ more efficiently than the internal consequence-based $\mathcal{EL}$-classifier that deals with $\mathcal{M}$ alone.

Apart from exposing some limitation to the unconditional reliance on the benign effect of modularity for reasoning, we believe that our work around pathological modules might be helpful in other ways. Firstly, we have developed a toolkit that allows reasoner developers to identify ontologies for which the reasoner behaves in a sub-optimal way. Understanding the reasons for the pathological behaviour might lead developers to isolate broken optimisations, causing them to introduce improved case distinctions (in the previous example, improve the test that determines whether an ontology or a module should be processed by the internal EL-reasoner) or to repair the optimisation. Secondly, we might learn something that can help us to develop better heuristics to lead reasoners into more benign branches in a non-deterministic setting. Thirdly, we might learn

[11] As algorithmic non-determinism can be effectively removed by fixing the order in which for example OR branches are explored it does *not necessarily* cause stochastic behaviour.

something about what makes reasoning hard. While the design of the experiments in this chapter is not targeted at understanding pathological cases, they do provide a first step by isolating interesting cases that are worthy of further investigation and hints on where to look. For example, it might be interesting to understand why the *protective effect* of modularity was so extreme (between 2 and 14 fold decrease in hardness between subset and module) in the four cases shown in Figure 5.8.

5.6.1 Methodological Reflection

In retrospect, Experiment 5.2 was sufficient to reject the original *benign module conjecture*. In essence, our sampling strategy was rooted in the assumption that *pathological modules are more likely to occur in performance-heterogenous ontologies*. Given the low success rate of the strategy (0.34%) and considering the slightly higher success rate of sampling from "real" modules (0.43%), we have to retract this belief. There are, however, a number of reasons why the method has some benefits. Firstly, we could observe the *protective effect* of modularity on hard subsets. The protective effect reduced the classification time of hard subsets significantly below the classification time of its parent ontology in 89% of the cases; slightly worse than expected. Secondly, it was *more effective* to search for hard modules given a set of hard subsets than merely classifying as many realistic modules as practical: While the "success rate" for encountering pathological modules among the Chainsaw and MORe modules was only around 0.64% (including the ones that were insignificantly pathological), it was almost 10.15% using the targeted search. The positive effect of that however is severely diminished by the the fact that we first had to determine the hard subsets, which cost about as many classifications as the classification of all MORe and Chainsaw modules in our sample. If one is interested to find pathological modules, the only other good argument for the first technique is the huge overhead induced by computing the Atomic Decomposition; sometimes more than 12 hours for a single ontology. Paths can obviously be computed in small fractions of that time.

The choice of sampling paths over merely sampling randomly of a hard proportion does not seem convincing in retrospect. We initially hoped to gain some insights also into the question of what makes a subset hard, but fell short of this goal. However, path sampling did reveal enough hard proportions for our investigation, and we did not see the need to repeat the experiment with purely random

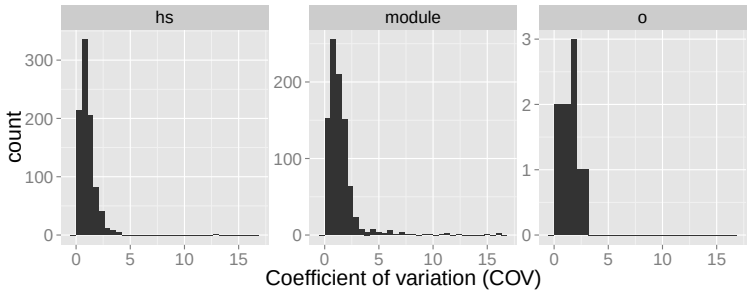


Figure 5.11: Histogram showing the distribution of the coefficient of variation (in %) of classification time across runs (of the same ontology or subset), broken down by type (hs: hard subset, mod: module, o: ontology).

subsets of a particular size region. Moreover, just being aware of the performance patterns a path exhibits, for example the very odd, bell-shaped case of FaCT++ on GO that can be seen in Figure 5.4, will provide food for thought and future experiments.

Another point of criticism might be the choice of 5% as a threshold for determining a *significant* pathological module. In retrospective, we can see that this threshold was a good choice. In Figure 5.11 we can see that only a few (1.54%) classifications vary by more than 5%, most of which are modules.

Reporting on OWL Full ontologies may appear pointless. We did feel however that, as long as we did not attempt to generalise conclusions about results obtained from OWL Full ontologies to OWL DL ontologies, merely studying them a bit better is of general interest. For example, we learned that OWL Full paths are often significantly non-monotonic, and the majority of pathological cases (as a result of our targeted search strategy) are caused by OWL Full ontologies.

Reasoner developers tune towards easily accessible ontologies and ontology corpora. This could mean that pathological behaviour is less prevalent in ontologies from BioPortal. It remains to be seen in the future whether the observation holds true for other ontologies found on the web.

Lastly, the fact that the classification of the real modules was reduced to a single run should be seen as a small threat to internal validity. While the overall ratios should not be effected too much, individual observations may differ sometimes due to measurement variance; compare also the variation results from Experiment 5.1.

5.7 Summary of Key Observations

We briefly summarize the key observations from our experiments:

- Significantly pathological modules exist but are rare: 0.34% as a result of our targeted search strategy and 0.43% as a result of classifying MORe and Chainsaw modules.
- 99.66.% of all hard subsets $\mathcal{HS}$ found by our targeted search strategy could be *easified* by taking their corresponding module $\mathcal{M}_{\widetilde{\mathcal{HS}}}$ (protective effect of modularity).
- 5.3% of all ontology reasoner pairs in the sample exhibit pathological behaviour for MORe and Chainsaw modules.

Chapter 6

Subsumption Test Hardness and Modularity

One of the core intuitions we explore as part of this thesis is that modularity reduces subsumption test hardness, see Section 3.4.1. The three main goals of this chapter are as follows:

1. Systematically investigate the role of subsumption testing in Description Logic classification in order to:

2. isolate ontologies for which subsumption test hardness reduction is potentially relevant and finally,

3. empirically establish potential threats and benefits of modularity on subsumption test hardness.

We conduct a survey across BioPortal measuring all tableau subsumption tests fired during classification, present a novel approach to robustly determine the effect of modularity on subsumption tests and apply it to a subset of BioPortal.

6.1 Definitions and Models

The phenomenon under investigation is **subsumption test hardness in the context of classification**. In the following, unless stated otherwise, a reasoner is a traversal/(hyper-) tableau style reasoner. A **subsumption test** occurs when the reasoner attempts to determine whether $\mathcal{O} \models A \sqsubseteq B$ by means of the underlying calculus. For OWL 2 DL, this calculus is usually based on some form of model construction, refutation based technique (such as tableaux). Most reasoners have elaborate optimisations designed to avoid engaging the core engine (most importantly traversal algorithms, see Section 2.3.2) in addition to intra-engine optimisations. The **subsumption test hardness** is the time it takes

to compute the answer, see Section 4.4.2. In this thesis the answer to a test is either yes or no. Note however that, for any given implementation, (1) more than just a binary answer may be computed and provided (e.g., pseudo-models [HMT01] may be constructed and cached) and (2) no guarantee is given that the answer is correct (i.e. the reasoner might be buggy). **In the context of classification** this means that we are not exploring individual "cold" tests, i.e. letting the reasoner decide whether $\mathcal{O} \models A \sqsubseteq B$ for any A, B from outside the classification process, because we want to understand the contribution of subsumption testing to classification as a whole, with all the optimisations involved. Note that we only measure actual subsumption tests, *omitting* most importantly the *initial consistency check*, which usually involves a call to the tableau engine. This is very important when interpreting subsequent results, especially the numbers of ontologies for which no subsumption test was recorded. It is also possible that the model generated during the consistency test is used to derive known (non-)subsumptions. For a detailed description of the process of classification see Section 2.3.2. We call *easyfication* the process of making subsumption tests easier, and *avoidance* the process of skipping a particular test, particularly through optimised traversal algorithms, see Sections 3.4.1 and 3.4.2.

Our **model of subsumption test hardness** with respect to sub-modules is based on the following intuition: given a subsumption test $ST(A, B, \mathcal{O})$ with $\mathcal{O} \models A \sqsubseteq B$, it should be the case that for every two modules $\mathcal{M}_1 \subset \mathcal{M}_2 \subseteq \mathcal{O}$ with $A, B \subseteq \widetilde{\mathcal{M}}_1$, the time it takes to compute $ST(A, B, \mathcal{O})$ is equal to the time it takes to compute $ST(A, B, \mathcal{M}_1)$ and $ST(A, B, \mathcal{M}_2)$ (if we ignore the overhead involved in determining (ir)relevant axioms); see Section 3.4.1.

Let $\mathsf{time}(X)$ be the time it takes for a process X to finish, $\mathcal{R}$ a reasoner, $\mathcal{M}_1, \mathcal{M}_2$ modules with $\mathcal{M}_1 \subset \mathcal{M}_2 \subseteq \mathcal{O}$ and $A, B \subseteq \widetilde{\mathcal{M}}_1$. Consider the possibilities:

1. $\mathsf{time}(ST(A, B, \mathcal{M}_1, \mathcal{R})) < \mathsf{time}(ST(A, B, \mathcal{M}_2, \mathcal{R}))$: This is what the model predicts since the search space gets more complex as we add information. We call this case *expected*.

2. $\mathsf{time}(ST(A, B, \mathcal{M}_1, \mathcal{R})) \approx \mathsf{time}(ST(A, B, \mathcal{M}_2, \mathcal{R}))$: This case is reasonable if the implementation can (cheaply) recognise that the problem $\mathcal{M}_2 \models A \sqsubseteq B$ can be restricted to $\mathcal{M}_1 \models A \sqsubseteq B$. A naive modular reasoner can potentially achieve this simply by extracting $\mathcal{M}_1$ from $\mathcal{M}_2$ and reasoning

over $\mathcal{M}_1$. Note that this might hurt overall classification time as we add in extraction overhead. We call this case *optimal*.

3. $\mathsf{time}(ST(A, B, \mathcal{M}_1, \mathcal{R})) > \mathsf{time}(ST(A, B, \mathcal{M}_2, \mathcal{R}))$: By our model, this case is *pathological* as the, in principle, harder (or equally hard) problem turned out to be easier. Somehow, the extra information makes the reasoner do better in spite of being strictly irrelevant to the problem at hand.

A possible explanation of the pathological case is that the implemented calculi are inherently non-deterministic, different choices can produce wildly different behaviour, and implementations make choices based on fallible heuristics. If the heuristics are sensitive to irrelevant information, then the effect of that irrelevant information might be to induce a significantly better choice by luck. Consider testing the satisfiability of a disjunction $C \sqcup D$, and that it is satisfiable because D is satisfiable and C is not. Obviously, we will typically do worse if we choose to explore D before C as determining the unsatisfiability of D is not necessary. Suppose our disjunction selection heuristic is length of the sub-expression and $|C| < |D|$. This is ok, and $time(SAT(C \sqcup D)) \approx time(SAT(C))$. Now, suppose we add a bit of information to C to get a C' such that $|C| < |C'| < |D|$. Now $time(SAT(C' \sqcup D)) \approx time(SAT(C'))$. We're a bit worse off, but as expected. Now, suppose that $|D| < |C|$. $time(SAT(C \sqcup D)) \approx time(SAT(D)) + time(SAT(C))$. If $time(SAT(D)) >> time(SAT(C))$ our heuristic made a very detrimental choice. But suppose we extend D such that $|C| < |D'|$ and the information added was completely irrelevant to C. Now, even though $time(SAT(D)) < time(SAT(D'))$, $time(SAT(C \sqcup D')) \approx time(SAT(C))$. That way, we can see how irrelevant information can interact beneficially with a heuristic.[1]

Moreover, depending on the form of the stochasticity, we might have highly variable time between runs of the very same module/reasoner pair. Consider for example the above optimisation, but instead of using sub-expression length as the heuristic, we replace it by random selection, or selection based on hash set traversal (no lexical order). In this case, we would, in some runs, get low values for $time(SAT(C \sqcup D))$, and, in other runs, high ones. Thus, any witness to a pathological case might merely be because *in that run* $\mathcal{R}$ was unlucky with respect to $\mathcal{M}_1$ and/or lucky with respect to $\mathcal{M}_2$. This potential lack of stability induced

[1] Note that this heuristic is not necessarily a bad one: it is very cheap, easy to implement, intuitive and likely to get the right option in lots of cases.

by stochasticity (rather than "mere" measurement error) makes it difficult to explain module-varying behaviour.

6.2 Empirical Characterisation

In this study, we investigate two research questions as described in the following. The first one aims at determining the relevance of subsumption testing for classification in general.

Research Question 6.1. *What is the relationship between subsumption test hardness and ontology classification time in practice?*

How much time does a reasoner spend on "real" reasoning, compared to preprocessing and traversal? If subsumption testing is significant to an ontology's classification for some reasoner, how is the time distributed across tests? Are there a few "killer" tests or do numerous easy tests dominate? Is difficulty randomly distributed across positive and negative tests? How many tests are done, and how effective are reasoners at avoiding tests? How much does the answers to these questions depend on the particular implementations?

Research Question 6.1 is important to understanding reasoning in general and in the design of modularity oriented procedures. For example, if it is typically the case that the total time spent doing subsumption tests constitutes only a small fraction of the overall classification time, the importance of modularity for making tests easier or avoiding them is diminished (but there might still be other beneficial effects from modularity).

In order to judge the impact of subsumption tests on classification performance, we draw on three different metrics.

- Overall classification time (OCT): This is the overall time it takes the reasoner to perform all stages, from pre- to post-processing (see Section 4.1).[2] This measure constitutes the upper bound for any gain through modularity. A very low absolute value may indicate that there is no need and space to further optimise.
- Sum of subsumption tests (SST): the sum of all times of tests triggered during a single classification run.

[2]Concretely: everything that happens from reasoner creation to finishing the classification using the OWL API.

- Hardest subsumption test (HST): the duration of the hardest test triggered during a single classification.
- Sum of subsumption tests to overall classification time ratio (SST/OCT): This tells us something about how much time the reasoner spends in the context of the tableau engine. A large value (close to 1) can suggest a need for optimising test avoidance as well finding more efficient ways to determine subsumption. A low value (close to zero) renders attempts to improve the performance of the tableau engine irrelevant. Note that by itself, the number does not directly imply the potential applicability of modular techniques, because modular techniques can in principle be beneficial in more ways than test avoidance or easyfication (for example by facilitating concurrent classification, or perhaps more efficient consequence-based reasoning). We say that subsumption testing has a *strong impact* on classification time if it accounts for more than 40% of the OCT. A medium impact is defined between 20% and 40% and a small impact between 0% and 20%.[3]
- Hardest subsumption test to overall classification time ratio (HST/OCT): This tells us something about the complexity of the reasoning problems inside an ontology. A large value (close to 1) means that a single test dominates the entire classification time. This raises the question whether it can be made easier or even be avoided altogether using modular techniques.
- Subsumption test count (STC): The number of subsumption tests triggered during classification. This number can be used to estimate the effectiveness of "normal" traversal algorithms (by comparing them against the n^2 and $n * log(n)$ upper bounds). Very low counts indicate that modularity might not be effectively usable to avoid further tests.

The second research question is of central importance for modularity-based classification:

Research Question 6.2. *How is a reasoner's subsumption test performance sensitive to modularly irrelevant axioms? In other words, is the behaviour of current reasoners expected, optimal, or pathological?*

If the behaviour is typically *expected* (see previous section) and we are dealing with ontologies for which subsumption testing has strong impact, then there is a clear opportunity for explicitly module sensitive procedures and optimisations.

[3]This classification is somewhat arbitrary, but it helps to isolate cases for which modular techniques for tableau test avoidance and hardness reduction are inapplicable.

A key sub-question here is whether the variance of reasoner performance between runs is sufficient to distinguish between stochastic variable performance and *module sensitive* variable performance. This is important in order to judge how reliably we can trace a single subsumption test through different sub-modules of an ontology, and may also give a warning sign for non-determinism, for example in the case that a test appears or disappears given a particular ontology-reasoner pair across runs.

We will address the problem of measurement stability mainly by (1) looking at the *coefficient of variation* (COV) of subsumption test hardness, see Section 4.5 for a detailed discussion, across different runs and (2) isolating cases where the classification time was potentially influenced by (obvious) stochastic effects. We use *varying number of tests triggered* across multiple runs as a first lower bound to label ontology-reasoners pairs as influenced by stochastic effects.

6.3 Experimental Design

We have conducted two separate experiments, each addressing one of our two research questions:

Experiment 6.1. *The characterisation of subsumption test hardness in the context of classification across BioPortal, addressing Research Question 6.1.*

Experiment 6.2. *The in-depth analysis of a subset of BioPortal for exploring the effect of modularity on subsumption test hardness, addressing Research Question 6.2.*

We conducted our study on the BioPortal snapshot described in Section 4.6.3 on Machine cluster 4.1 (Section 4.4.3).

6.3.1 Experimental Pipeline

Landscape of Subsumption Test Hardness

For the first experiment, we executed for each reasoner a single run across the entire corpus, with a timeout of 60 minutes per run. Due to technical details, the timeout constituted a lower bound and might not have been triggered until some minutes later. Note that we included every ontology in the corpus, including the ones not strictly in OWL 2 DL (53), but isolate them in the analysis.

The reason for that is that these ontologies do form part of the landscape, and reasoners are used on them. The main sources of violations were uses of reserved vocabulary (37% of all violations across the corpus), illegal punning (32%) and uses of datatypes not on the OWL 2 datatype map (11%).[4]

Effect of Modularity

For the second experiment, we selected a set of ontology-reasoner pairs for which, according to the results of the previous experiment, at least one subsumption test was measured that was harder than 100 milliseconds. This bound is set for convenience: it results in a nice sample size, it is easy to memorise and it is clearly non-trivial. Because of the various claims we have with respect to modules, we also excluded ontologies that do not fall under OWL 2 DL. *Runtime limitations* forced us to exclude the NCIt from the sample, due to the extreme number of measured subsumption tests (JFact 751,907 tests, Pellet 461,831, FaCT++ 605,481). In order to answer Question 6.2, we classified tests by analysing how modularity affects their hardness. First we identified all super and sub-module combinations $\mathcal{M}_1, \mathcal{M}_2$ as follows: We obtained random cumulative subsets from the ontologies in our narrowed down sample, similar to Gonçalvez et al. [GPS12], with 16 slices. We use the same approach discussed in Section 5.2.1. From the signature of each subset sampled, we obtained the $\bot$-locality module using the OWL API module extractor. Module properties ensure, given two subsets S_1, S_2 with $S_1 \subseteq S_2$, that $\mathcal{M}_{\widetilde{S_1}} \subseteq \mathcal{M}_{\widetilde{S_2}}$ [SSZ09]. The module of $\frac{16}{16}$th, $\mathcal{M}_{\widetilde{\mathcal{O}}}$, corresponds to the whole ontology. We call this nested set of modules a **path**. Note that the modules are on average 40% larger than their respective subsets, which will give us a good sample of relatively large modules with hopefully hard subsumption tests. Each of the modules obtained was classified three times (i.e., three independent runs) by each reasoner. Given a path $\mathcal{M}_1 \subseteq \mathcal{M}_2 \subseteq ... \subseteq \mathcal{M}_n$, we call $\mathfrak{P}$ the set of all pairs $\mathcal{M}_i, \mathcal{M}_j$ with $i < j$. Given a pair $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \in \mathfrak{P}$ and a reasoner $\mathcal{R}$ we compare $CT(\mathcal{M}_1, \mathcal{R})$ with $CT(\mathcal{M}_2, \mathcal{R})$ and changes in subsumption test hardness: $ST(A, B, \mathcal{M}_1)$ with $ST(A, B, \mathcal{M}_2)$.

Every pair of measurements has a *tendency*, a *magnitude* and a *degree of stability*, see Table 6.1.

We call the tendency *easier* if a test is easier in the super-module than in the sub-module (potentially pathological), *harder* the respective reverse, and *neutral*

[4]For more details on the violations, see Section 4.6.3

Table 6.1: Dimensions of subsumption test hardness *change under modularity*.

Feature	Value	Measurement
Tendency	Easier	Mean hardness change < -5%
	Harder	Mean hardness change > 5%
	Neutral	Absolute mean hardness change < 5%
Magnitude	High	Absolute mean hardness change > 50%
	Medium	Absolute mean hardness change 5% - 50%
	Low	Absolute mean hardness change < 5%
Stability	Clear cut	All measurements same tendency
	High	Overlap of ranges of measurements < 10%
	Low	Overlap of ranges of measurements > 10%

if the mean measurement difference does not differ by more than 5% between sub- and super-module. *High magnitudes* are changes above 50% (the test is more than 50% harder/easier in the super-module compared to the sub-module), medium magnitudes are changes between 5% and 50% and low changes are absolute changes below 5%. An effect can be of three *degrees of stability*: *clear cut*, *high* or *low*. Given a subsumption test $ST(A, B, \mathcal{O})$ that occurs in two modules $\mathcal{M}_1, \mathcal{M}_2$ with $\mathcal{M}_1 \subset \mathcal{M}_2$, and two sets of measurements $ST(A, B, \mathcal{M}_1, \mathcal{R}, i)$ and $ST(A, B, \mathcal{M}_2, \mathcal{R}, j)$, we call the change stability:

- *Clear cut,* if either
  - for each measurement $M_i \in ST(A, B, \mathcal{M}_1, \mathcal{R}, i)$ and $M_j \in ST(A, B, \mathcal{M}_2, \mathcal{R}, i)$ we have that $M_i < M_j$ or
  - for each measurement $M_i \in ST(A, B, \mathcal{M}_1, \mathcal{R}, i)$ and $M_j \in ST(A, B, \mathcal{M}_2, \mathcal{R}, i)$ we have that $M_i > M_j$.
- *High*, if the overlap of the ranges of $ST(A, B, \mathcal{M}_1, \mathcal{R}, i)$ and $ST(A, B, \mathcal{M}_2, \mathcal{R}, j)$ is less than 10% of the range of $ST(A, B, \mathcal{M}_1, \mathcal{R}, i) \cup ST(A, B, \mathcal{M}_2, \mathcal{R}, j)$
- *Low*, if the stability is neither clear cut nor high.

Note that cases of *neutral tendency* have high stability if both sets of measurements have a variation coefficient of less than 5%. The example in Figure 6.1 shows a clear cut hardness change from module $\mathcal{M}_1$ to $\mathcal{M}_2$, but one with low stability from $\mathcal{M}_2$ to $\mathcal{M}_3$. We sometimes refer to changes of more than 5% as significant.

Figure 6.1: An example for multiple measurements taken for a single subsumption test across three modules $\mathcal{M}_1 \subset \mathcal{M}_2 \subset \mathcal{M}_3$.

6.4 Results

Percentages in this section are subject to uniform rounding. The measures used in this section (OCT, HST, SST) were described in Section 6.2. We group subsumption test hardness into the following bins: Very Hard ($>$ 100 seconds), Hard ($>$ 10 sec, $\leq$ 100 sec), Medium Hard ($>$ 1 sec, $\leq$ 10 sec), Medium ($>$ 100 ms, $\leq$ 1 sec), Medium Easy ($>$ 10 ms, $\leq$ 100 ms), Easy ($>$ 1 ms, $\leq$ 10 ms), Very Easy ($>$ 100 μs, $\leq$ 1 ms), Trivial ($\leq$ 100 μs).

Out of the 1,320 attempted classification runs (4 reasoners and 330 ontologies), 1,109 (84%) completed successfully. Of the 330 ontologies, 240 ontologies (81%) were dealt with by all four reasoners within the 60 minute timeout, another 24 by 3 reasoners, 28 by 2, 21 by just 1 and 17 by none of the four reasoners, see Table 6.2. Since we have considered OWL Full ontologies in this particular survey, we present the numbers of successfully classified ontologies broken down by OWL profile category and whether they triggered subsumption tests.[5] Table 6.2 also serves as a binning for further analysis later on, so it makes sense to study it carefully before moving on. Note that out of the 17 ontologies no reasoner classified within the timeout, none fell under OWL 2 EL, QL or RL.

Test	All reasoners successful				Some reasoners successful				All Unsuccessful			
All	240				73				17			
Profile	All	DL	Prof	Full	All	DL	Prof	Full	All	DL	Prof	Full
All Prof.	240	75	146	19	73	37	8	28	17	11	0	6
No test	152	13	136	3	25	4	8	13				
Some test	41	23	10	8	48	33	0	15	17	11	0	6
All test	47	39	0	8	-	-	-	-	-	-		-

Table 6.2: Binning of all 330 ontologies by success category and test category.

FaCT++ completed 268 ontologies in total (81%), HermiT 284 (86%), JFact

[5]For example, 13 DL ontologies were classified by all four reasoners (*all successful*) but only some of the reasoners triggered a subsumption test.

270 (82%) and Pellet 287 (87%). Reasons for failure include hitting the timeout, unsupported datatypes, and ontology inconsistencies. Table 6.3 shows a detailed account of that; note however that a grouping of reasons for failure by Java exception is not very informative of the actual reasons of failure. It is very likely that FaCT++ for example throws a ReasonerInternalException when encountering an unsupported datatype. Pin-pointing the exact source for each failure for each reasoner is out of the scope of this thesis.

	FaCT++	HermiT	JFact	Pellet
illegalargument	1	16	1	1
unsupportedoperation	0	0	0	1
owlreasonerruntime	10	0	0	0
reasonerinternal	23	0	29	0
nullpointer	0	0	5	0
arrayindexoutofbounds	0	0	0	1
concurrentmodification	0	0	0	1
inconsistentontology	3	2	0	7
malformedliteral	0	1	0	0
unsupporteddatatype	0	14	0	0
numberformat	0	0	1	0
timeout	4	3	2	24
unknown	21	10	22	8
success	268	284	270	287

Table 6.3: Detailed account of successes and failures, as they were reported in the form of Java Exceptions. Unknown items are most likely those that had to be terminated by the test framework, thereby not leaving an explanation of failure.

In order to improve our understanding of how different reasoners compare, we present in Table 6.4 the number of ontologies broken down by (1) which reasoner dealt with them and (2) which reasoners they caused to trigger a subsumption test. As (1) has a strong impact on (2), we decided to present for (2) only the break-down of the 240 ontologies that all four reasoners successfully classified. (2) will be discussed in more detail later on. Note that with respect to (1) there is almost no discernible pattern of reasoner factions (i.e. groups of reasoners that behaved in a similar fashion). A large proportion of all possible combination is represented (including unlikely dominant combinations such as JFact, FaCT++ and HermiT, given that the strongest faction, Pellet all by itself (and the fact that it has the most completed classifications of all four reasoners), appears to be a sign that Pellet is quite resistant to failure.

J	F	P	H	freq
X	X	X	X	240
				17
		X		13
X	X		X	9
X		X	X	8
X		X		7
		X	X	7
	X		X	6
			X	6
	X	X	X	5
	X	X		5
X			X	3
X	X	X		2
X				1
	X			1

J	F	P	H	freq
				152
X	X	X	X	47
			X	18
X	X			10
X	X	X		7
X	X		X	5
	X			1

Table 6.4: Left: Combinations of reasoners to successfully classify an ontology out of all 330 ontologies. Right: Combinations of reasoners to fire tests out of the 240 ontologies that all reasoners dealt with.

6.4.1 Role of Subsumption Testing in Classification

In the following, we describe our observations that help us in addressing Research Question 6.1. We break the main question down into the following sub-questions before we summarise the most important observations in our discussion.

1. **Which ontologies cause reasoners to trigger subsumption tests?** First, we will narrow down how many ontologies are affected, and what kinds of ontologies are affected. Then we will take a brief look at the differences between reasoners.

2. **What are real subsumption tests like?** We will see how many tests are generally fired and we will analyse them in terms of hardness and the differences between positive and negative subsumption tests, before we look in more depth at the differences between the four reasoners.

3. **What is the Contribution of Subsumption Test Hardness To Classification Performance?** We will take a look at the SST/OCT ratio (see Section 6.2), determine the effect of very hard tests and the potential gain involved making these tests easier or less numerous.

4. **What are the shared characteristics of ontologies with a high impact factor?** For those ontologies for which we determined that subsumption testing plays a big role, what are they like? Are there any obvious structural characteristics that these ontologies share?

5. **What is the performance of traversal algorithms?** In this ancillary question, we will look at how traversal algorithms fare against the $n*log(n)$ and naive n^2 upper bounds.

Which Ontologies Cause Reasoners to Trigger Subsumption Tests?

Our first observation is that 152 (**46.1%) of the ontologies were classified by all four reasoners and did not trigger a single subsumption test** [6], see Table 6.2. Another way to put it (from the perspective of successful classifications): only 33% of all ontology-reasoner pairs in the set of successful classifications involved one or more calls to the tableau engine.[7] *All four* reasoners fired tests in 47 of the ontologies. 136 ontologies caused at least one reasoner to conduct a subsumption test. Out of these 136 ontologies, only 10 ontologies fall under one of the profiles, all of which are pure OWL 2 EL (i.e., they neither fall under OWL 2 QL nor OWL 2 RL). The remaining 144 ontologies falling under one of the profiles did not trigger any tests at all. 70% of the above 136 ontologies are Pure DL, that means of considerable expressivity.

Differences across reasoners. Overall, FaCT++ did not test in 177 cases (66% of successful classifications), HermiT in 180 (63%), JFact in 182 (67%), and Pellet did not fire a subsumption test during 209 (73%) successful classifications.

To get a picture about the agreement between the reasoners on whether calls to the tableau engine are required at all, we will zoom in on the 240 ontologies that all four reasoners successfully dealt with. Table 6.4 (right side) shows how reasoners differ in opinion whether tests are necessary. The two largest factions in cases of disagreement are HermiT all by itself (18 ontologies) and JFact/FaCT++ (10 ontologies). The second faction is perhaps explained by the architectural similarity between FaCT++ and JFact, see Section 4.7.2. 63% of the 240 ontologies did not trigger a test by any of the reasoners. In 8% of the ontologies, only one reasoner (mainly HermiT, see above) triggered a test, in 4% two reasoners, in 5% three reasoners and in 20% all four reasoners. Note that all cases for which there

[6] The reasons for this are still unknown, and investigating them is part of future work.

[7] It is important to remember that this excludes the initial consistency check, see Section 6.1.

is no agreement on whether tests are necessary or not essentially indicate missed opportunities for optimisation (at least one reasoner managed to classify without firing a test).

Table 6.5 shows how these 240 ontologies are distributed across the size bins (for the definition of the size bins, see end of Section 4.6.3). The main observation to take from that is that most disagreements (proportionally) happen among the medium and large ontologies (between 100 and 10,000 axioms). We define *agreement* as the cases where either all reasoners or no reasoners fired a test, and *disagreement* the respective reverse.

$\lvert\mathcal{R}\rvert$	empty	very small	small	medium	large	very large	huge
1	0	0	2	11	5	1	0
2	0	0	0	3	7	0	0
3	0	0	1	8	2	1	0
4	0	0	1	24	15	6	1
0	0	2	21	58	52	17	2

Table 6.5: Contingency table showing ontology size to number of reasoners ($|\mathcal{R}|$) to fire one or more subsumption tests. The top three rows reflect disagreement (see text), the bottom two agreement between the reasoners.

Table 6.6 shows how the distribution of ontologies with respect to their expressive power (or a proxy thereof) and the number of reasoners firing tests while dealing with them. Only a handful of ontologies that fall under the OWL 2 RL, EL or QL profiles (10) force some reasoner to trigger a test. All 10 cases are caused by JFact and FaCT++ (always both). For 18 of the 19 cases that all reasoners successfully dealt with and only one of the four reasoners firing a test, that reasoner was HermiT (one FaCT++ case).

$\lvert\mathcal{R}\rvert$	OWL Full	Profiled	Pure DL	All
0	3	136	13	152
1	7	0	12	19
2	0	10	0	10
3	1	0	11	12
4	8	0	39	47

Table 6.6: Contingency table showing ontology OWL profile bin to number of reasoners ($|\mathcal{R}|$) to *fire a test*.

$\mathcal{R}$	Min.	Q1	Med.	Avg.	Q3	Max.	$\lvert\mathcal{ST}\rvert$+	$\lvert\mathcal{ST}\rvert$-
F	2	46	71	7,519	111	2,352,000	24,286	905,011
H	48	418	481	17,390	570	198,900,000	1,911	88,387
J	1	48	88	1,127	169	45,920,000	28,103	1,100,972
P	23	175	246	825	365	35,060,000	634	522,592

Table 6.7: Subsumption test hardness: Descriptive Statistics (unit μs), number of positive ($\lvert\mathcal{ST}\rvert$+) and negative ($\lvert\mathcal{ST}\rvert$-) tests, by reasoner $\mathcal{R}$.

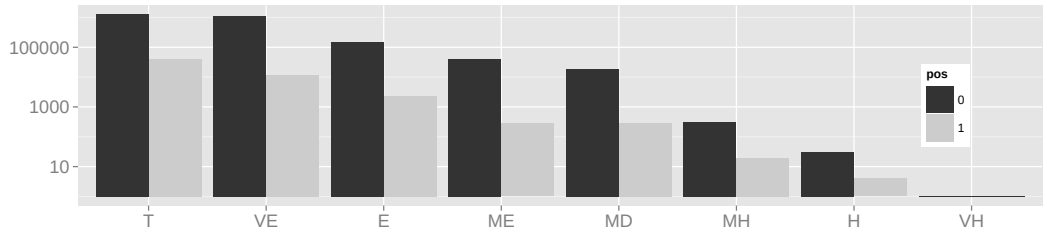


Figure 6.2: Counts of subsumption tests for each hardness bin (log scale), distinguished by positive (1) and negative (0) tests.

What are Real Subsumption Tests Like?

Across all 1,109 successful classifications we measured 2,671,896 subsumption tests, 54,934 out of which turned out positive (2.06%) and 2,616,962 out of which were negative (97.94%), see Table 6.7. Positive tests account for only between 0.12% (Pellet) and 2.61% (FaCT++) of the overall number of tests (HermiT 2.12%, JFact 2.49%). This low ratio is not surprising, see Section 2.3.2, but we do not currently know why the ratio for Pellet is much lower than the one of the others, despite their architectural similarities. Subsumption test hardness varies widely: while most subsumption tests are easy (e.g., half of all tests take less than 481 μs for HermiT and less than 71μs for FaCT++), the hardest ones take over 3 minutes.

Figure 6.2 shows the distribution of subsumption tests across the hardness bins. Keeping in mind that the figure is presented with a logarithmic scale, we can see that by far the majority of tests are negative/trivial or negative/very easy, i.e. 90.18% of all measured test take less than a millisecond.

To get a better picture of the distribution of tests, see Figure 6.3. A striking feature of HermiT appears to be that negative tests are densely centred around approximately half a millisecond, while all the other reasoners appear to have a larger spread of negative test hardness. As a reminder, positive subsumption tests correspond to negative satisfiability tests, which have often assumed to be

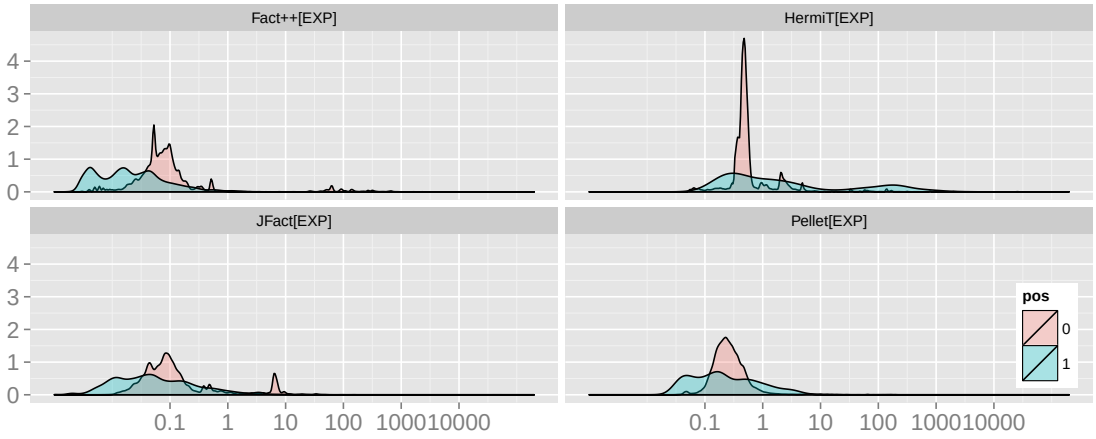


Figure 6.3: Kernel density plot of subsumption tests for each hardness bin (x:log scale, milliseconds), distinguished by positive (1) and negative (0) tests. Subsumption tests across entire experiment.

harder because in the worst case, all branches need to be explored to verify the unsatisfiability. Another observation to note is that FaCT++ and JFact find a number of negative tests hard (and no positive ones), while HermiT triggers some tests that are hard and turn out positive, and few hard negative ones.

In order to better understand how reasoners differ in terms of subsumption test hardness, we will look at the *39 pure DL ontologies* processed by all four reasoners in more detail. The distribution of subsumption test hardness for those ontologies is shown in Figure 6.4. Only HermiT fires a handful of positive tests harder than 100 ms. In terms of negative tests, only Pellet and HermiT have tests harder than 1 second, and only a handful. Another observation to take away is that towards the very easy part of the plot (less than 0.01 ms), we find for FaCT++, JFact and Pellet more positive than negative tests. For these three, negative tests also seem to be approximately (log-)normally distributed. Whether the differences for HermiT are due to the architectural differences is up for further investigation.

To answer the question whether positive tests or negative tests are generally harder, the choice of the measure of central tendency is crucial. As we can see in Figure 6.4 the distribution of test hardness is in most cases not even log normal, in some cases they appear even multi-modal (see positive FaCT++ tests). The high skew renders the mean a quite deceptive tool to compare the hardness of positive and negative tests. In terms of median, which is quite insensitive to outliers but more applicable for these kinds of distributions, we cannot learn

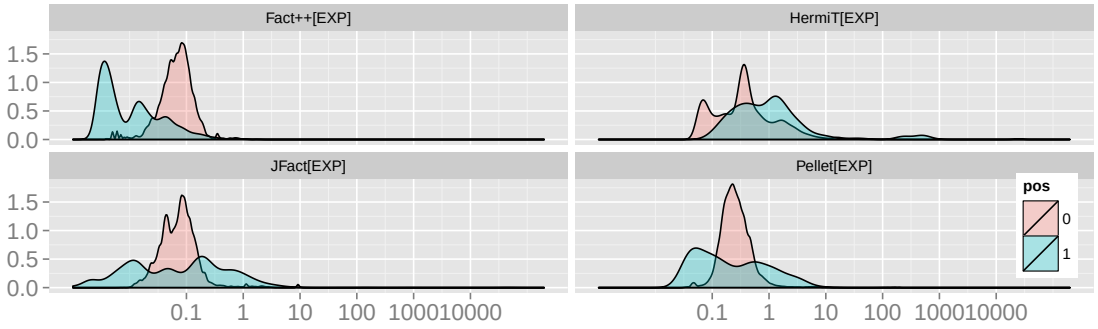


Figure 6.4: Kernel density plot of subsumption tests for each hardness bin (x:log scale, milliseconds), distinguished by positive (1) and negative (0) tests. Subsumption tests across 39 OWL 2 DL ontologies with tests triggered by all four reasoners.

much. For FaCT++, tests resulting in non-subsumptions are generally harder than positive tests (10 times on average, using median), for HermiT 2.6 times and for Pellet 1.5 times. JFact, surprisingly, finds positive tests 1.05 times harder than negative tests. As these aggregations are largely dominated by the fast tests, it makes sense to take a closer look at the harder ones. Since hardest test measurements are quite sensitive to experimental error, we focus our attention on the 90th quantile. Here, the picture appears almost reversed. For FaCT++, negative tests are still 2.3 times harder than positive ones. For HermiT *positive* tests appear 2.03 times *harder* than negative ones, for Pellet 3.2 times and for JFact 5.5 times. One interesting observation is the striking similarity between Pellet and JFact. Both appear to have a wide range of positive tests, and a large "pillar" of negative tests almost at the center of it, where there is also a small downward bulge from the positive tests.[8] As we described in Section 3.4.1, we expected negative subsumption tests to be typically harder. It is part of future work to explain why some reasoners find negative, and some reasoners positive tests harder; any attempts at an explanation given the current data would remain speculation.

To determine how prevalent individual hard reasoning problems are, it makes sense to group the ontologies in our corpus by the *hardest test* fired. Figure 6.5 shows the entire corpus binned by hardest test. The most important observation to make here is the rarity of ontologies with tests that take longer than a second (medium hard bin and above). The dominating cases are ontologies whose hardest

[8]To date, there is no explanation for this phenomenon.

Figure 6.5: Ontologies in each hardness category. The x-axis represents the number of ontology in each bin. Bin classification according to hardest subsumption test.

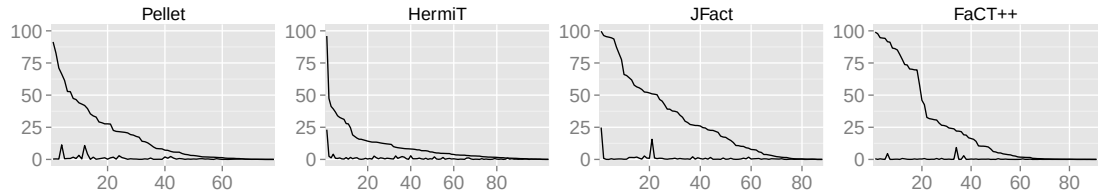


Figure 6.6: Impact of SST on classification time by reasoner in %. Low line: hardest individual test; high line: sum of all tests; x-axis: ontologies; y-axis: contribution in %. Note that x-axis values mean something different for each reasoner: The set of ontologies for which a given reasoner triggered subsumption test, ordered by ratio of SST to OCT.

tests range between 1 ms and 100 ms. Note that this observation of low hardness contradicts observations made by Gonçalvez et al. [GPS12]; many of the tests they measured were harder than 100 ms. However, satisfiability checks (in their experiments) were performed in a blackbox fashion from outside the process of classification, which may have introduced some overhead due to the particular implementation of the satisfiability method of a given reasoner.

What is the Contribution of Subsumption Test Hardness To Classification Performance?

In Figure 6.6 we show the contribution of all subsumption tests and the contribution of the hardest test to the overall classification time (OCT), broken down by ontology and reasoner.

While the sum of all subsumption tests dominates the OCT only in a few cases

(very few for HermiT, more for JFact and FaCT++), it occasionally accounts for more than 80%. Only 1 ontology has more than a 50% contribution of total SST on OCT for Hermit, 7 for Pellet, 19 for FaCT++ and 23 for JFact. Very rarely can we observe a single test accounting for more than 10% of the OCT. The maximum impact for a single test by FaCT++ is 9.2%, Pellet 11.3%, HermiT 23.1% and JFact 24.8%. According to our notion of *strong impact* (see Section 6.2) we count 3 for ontologies HermiT, 12 for Pellet, 21 for FaCT++ and 26 for JFact.

Trivial and very easy tests dominate by far in terms of number, but they are *not responsible for the majority of the impact* on OCT. Table 6.8 breaks down the overall impact of tests belonging to a particular hardness category across the entire corpus (first row), compared to the subset of 39 DL ontologies that all four reasoners processed (second row). For example, the sum of all negative (0) hard (H) tests across all 39 DL ontologies is 10.9 minutes. Negative medium (M) and negative medium-easy (ME) tests, i.e. tests between 10 ms and 1 sec, dominate performance by far (overall, first row). Interestingly, among the 39 pure OWL 2 DL ontologies that all four reasoners dealt with (second row), the dominating type of tests are hard negative tests (tests between 10 and 100 seconds). For a discussion on the hardness of non-subsumptions see Section 3.4.1.

T		VE		E		M		ME		MH		H		VH
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1.2	0.0	5.1	0.1	12.2	0.1	90.0	1.4	31.6	0.2	10.4	0.5	12.1	3.1	3.3
1.1	0.0	3.3	0.0	1.2	0.0	1.6	0.2	0.4	0.0	0.4	0.0	10.9	3.1	3.3

Table 6.8: This table shows the sum of all tests in each category in minutes. The first row accounts for all subsumption tests measured as part of this experiment, the second corresponds to the 39 pure DL ontologies measured by all four reasoners. The abbreviations T, VE,..,VH correspond to the hardness bins from Trivial to Very Hard, and the 0 or 1 indicates whether the test turned out to be positive (1) or negative (0).

An optimisation that reduced the mean hardness of subsumption tests by 50% would, taking into account only those ontologies for which tests were triggered, reduce the overall classification time on average by 14.2%. There are however differences between the reasoners: for JFact, the overall classification time could be reduced by 21.3%, for FaCT++ by 19.4%, for Pellet by 11.4% and for HermiT only by 5.8%. If we only take into account the 39 ontologies for which all four reasoners triggered tests, we would have a similar picture: the OCT would be

reduced on average by 16.6% for JFact, by 14.02% for FaCT++, by 12.03% for Pellet and by 6.9% for HermiT.

What is the Profile of Ontologies with a High Impact Factor?

Only 10 unique ontologies across the corpus cause triggering of *hard tests* (harder than 1 second) by any one reasoner, 8 of which are Pure DL (all beyond $\mathcal{ALC}$), and 2 of which are in OWL Full. Out of the 8 OWL 2 DL ontologies, 4 are only of medium size (between 100 and 1,000 axioms), 3 are large (between 1,000 and 10,000 axioms) and 1 ontology has more than 100,000 axioms. 7 of the 8 ontologies involve inverse roles, and all 8 ontologies involve role hierarchies (either languages $\mathcal{H}$ or $\mathcal{R}$).

Among the 38 ontologies for which at least one reasoner registered a *strong impact* of subsumption testing on the performance of the overall classification time, 30 were pure OWL 2 DL, 2 fell under OWL 2 EL (fairly inexpressive $\mathcal{ALE}$) and 6 were OWL Full. The ontologies are scattered across most size ranges: 4 have more than 10,000 axioms, another 20 more than 1000 and 14 ontologies have less than 1000 axioms. 33 of these 38 ontologies contain inverse roles, and 35 contain role hierarchies, 19 out of which contain more complex role-related modelling ($\mathcal{R}$). More than half contain nominals.

What is the Performance of Traversal Algorithms?

Current traversal algorithms appear to be mostly efficient, see Figure 6.7, in particular the ones implemented in Pellet and HermiT: they exceed the $N*\log(N)$ upper bound only once, and no reasoner comes even close to the naive N^2 upper bound (N being the number of class names in $\widetilde{\mathcal{O}}$). We discussed both of these upper bounds in Section 2.3.2. Exceeding the $N * \log(N)$ upper bound suggests that the reasoner was faced with a poly-hierarchy, i.e. an ontology involving concepts with multiple super-classes. Exploring the relationship between the number of triggered subsumption tests and the shape of the class hierarchy is part of future work.

Discussion

A first answer to Question 6.1 is that (tableau) subsumption testing does not contribute at all to classification time for a substantial number of ontologies. We

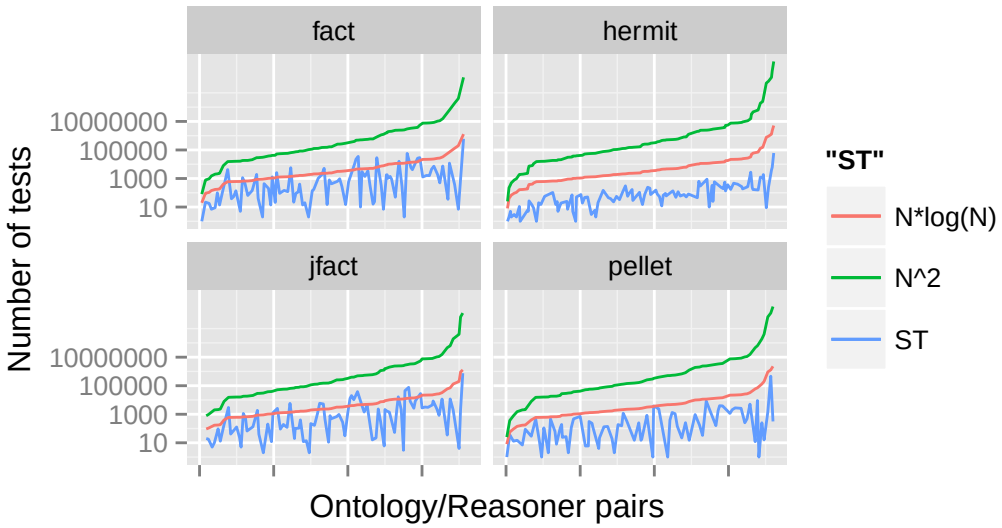


Figure 6.7: Subsumption tests carried out in relation to a naive N^2 upper bound and an $N \log(N)$ upper bound, ordered by N, the number of names in $\widetilde{\mathcal{O}}$ (y: log scale).

have established a *hard lower bound* for ontologies that do not involve subsumption testing at *46% of BioPortal*. This lower bound is even slightly naive because (1) there are most likely a number of ontologies that do not involve tests among the unsuccessfully classified ones and (2) only 33% of all ontology-reasoner pairs involved tests. Considering a theoretical union reasoner that always picks the reasoner that fires the least tests further reduces the need for subsumption test avoidance and easyfication considerably. The currently secured *lower bound* for ontologies actually requiring subsumption testing lies at *14%* (i.e. the 47 out of 330 ontologies for which all four reasoners triggered a test). Note that, while this might seem like a very low number, these might be the 50 or so ontologies in the world that are hard and *matter*, and thus worth optimising for. As a side note, the low numbers of tests for HermiT and Pellet can perhaps be explained by their internal alternative deterministic engines (for example internal $\mathcal{EL}$-reasoners), see Section 4.7.2.

It is quite interesting that only 10 out of those 146 ontologies that all reasoners processed caused at least one reasoner to fire a test—all of which are *pure* OWL 2 EL. Ontologies of the OWL 2 RL or OWL 2 QL family, or less expressive ontologies, did not cause any reasoner to actually fire a test. This suggests that for OWL 2 RL and OWL 2 QL ontologies at the very least, the application of

modular techniques must be strictly motivated by a different argument than test avoidance or test easyfication. Another potentially interesting observation is that ontologies involving hard tests generally seem to contain rich role-level modelling, most prominently inverses and role hierarchies.

Subsumption test hardness rarely has a strong impact on classification performance. According to our threshold of "strong impact" at 40% of the overall classification time, FaCT++ encountered impactful ontologies 7.8% of the time, JFact 9.6% of the time, Pellet 4.2% of the time and HermiT only in 3 out of its 284 successful classifications (just around 1.1%). This, and taking into account the low absolute potential performance gains as described above, creates a *case against modular reasoning techniques* motivating them the way we currently do (avoidance, hardness reduction). These results do not affect modular techniques motivated differently (like partial reclassification in incremental reasoning or expressivity reduction for partial classification by cheaper algorithms such as MORe).

The most important *threat to the validity* of the results presented in the previous section is the tight (albeit necessary) *timeout of 60 minutes*, a limitation that pervades most experiments presented as part of this thesis. It might well be that the 211 missing ontology-reasoner pairs all triggered subsumption tests, even hard ones, before they timed out, or would have triggered tests if they were not rejected for an unsupported datatype. In principle, these failed cases could create a very good case for modular reasoning. A second threat to validity is the *unavailability*[9] *of fine grained correctness* benchmarks. In an independent study [LMPS15] we revealed at least 9 ontologies in our corpus for which there was disagreement between the reasoners. Incomplete reasoning could potentially skew the measurements (test counts and duration) significantly. OWL 2 Full ontologies are not dealt with uniformly by all reasoners; essentially anything can happen when reasoning with them (for example dropping axioms). This is why the survey does not lend itself to comparisons of the performance of the reasoners directly (beyond what has been presented).

[9] At least up until the time of writing, see our recent work [LMPS15].

6.4.2 Sensitivity to Modularly Irrelevant Axioms

The previous sections were dedicated to studying monolithic traversal/(hyper-) tableau style classification across BioPortal, specifically for improving the understanding of the role of subsumption testing for OWL classification. In the following section, we will discuss Research Question 6.2. We will first describe general difficulties of measuring the "effect of modules" on subsumption test hardness, before we address the question and present our insights on *sensitivity to modularly irrelevant axioms*, or, in other words, the effect of modularity on subsumption test hardness, with respect to all four reasoners.

From the previous experiment, according to the process detailed in Section 6.3.1, 3 ontologies were selected for FaCT++, 4 for Pellet, 5 for JFact and 13 for HermiT for the following analysis (25 ontology-reasoner pairs). These ontologies have overall classification times ranging from 7 to more than 1200 seconds (median: 103.20, mean: 210.70). 16 modules were generated per ontology and classified, which led to *1,200* attempted classifications (3 runs per module, 25 ontology reasoner pairs) with a timeout of 60 minutes. Of these, 1,093 (91%) terminated successfully. By reasoner, that is 77.1% for FaCT++, 91.8% for HermiT, 98.4% for Pellet and 100% for JFact. Out of the 400 module classifications (16 modules, 25 ontology-reasoner pairs), we have 358 (89.5%) for which we have three successful classification runs. We exclude the remaining ones from the analysis.

Measuring the Effect of Modularity

There are two important factors that potentially threaten the internal validity of our results: (1) Experimental error (measurement variability caused by factors outside the program) and (2) stochasticity in the classification process. Both problems relate to different phenomena, but are usually not distinguishable from our experiment data alone.

Multiple runs of the same program do not usually lead to the same execution times. Experimental error (1) is a major cause for variance across timings. For example, system-level processes may kick in and add to the CPU load or varying times for memory allocation and garbage collection. Our measure for variability is (again) the *coefficient of variation* (COV), see Section 4.5. For each module-reasoner pair, we look at the variability of two distinct variables: overall classification time (OCT) and sum of all subsumption test times (SST).

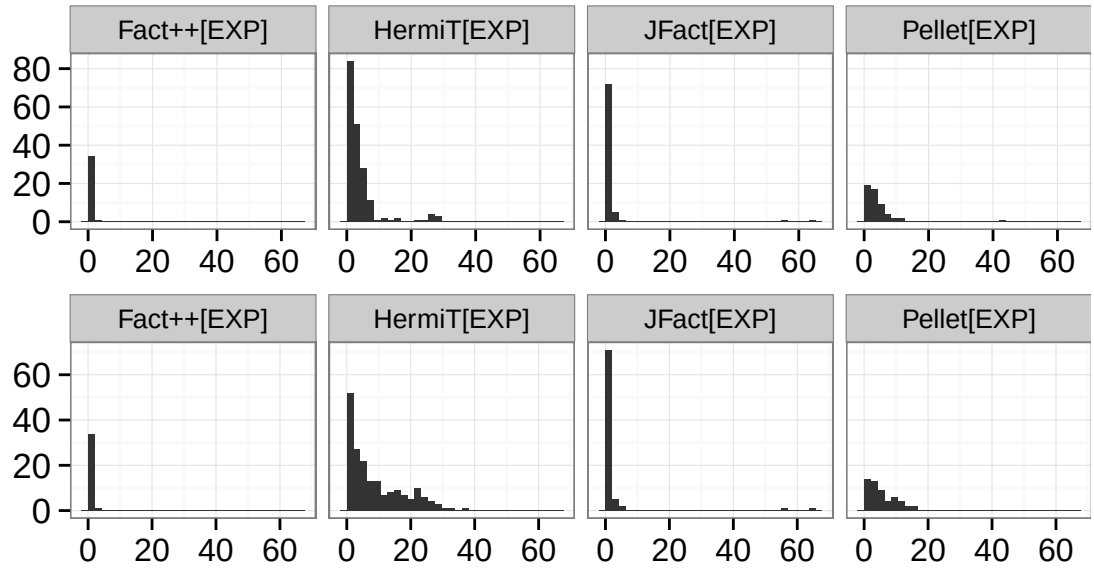


Figure 6.8: Histogram of COV, by reasoner. Top: OCT, bottom: SST

The OCT of only 3 out of 358 (0.84%) module-reasoner pairs (MRP) varies by more than 30%, of 12 (3.4%) MRP's (including the ones above 30%) by more than 20%, and of 19 (5.3%) MRP's by more than 10%. The module with the worst variation corresponds to a module taken from a $\frac{2}{16}$th of the Biotop ontology (727 logical axioms), classified by JFact (min=38.49 sec, max=194.22 sec). By reasoner, FaCT++ varies for OCT on average (median in brackets) by 0.79% (0.67%), JFact by 2.53% (0.88%), Pellet by 4.32% (3.12%) and HermiT by 4.15% (2.33%).

For SST, the reasoners vary as follows: FaCT++ 0.77% (0.66%), HermiT 9.02% (5.65%), JFact 2.61% (0.86%) and Pellet 5.43% (4.28%). A more detailed picture of the overall variation is given in Figure 6.8 for both OCT and SST.

Next we consider the variance of individual subsumption test hardness: across all 358 module-reasoner pairs, we measured the hardness of 2,438,983 distinct subsumption tests. As can be seen in Figure 6.9, the coefficient of variation is generally log10-normally distributed, but varies considerably across reasoners. On average, measurements deviate by as much as 13.22% and 13.96% for Pellet and HermiT, respectively, while measurements for JFact deviate by 3.5%, and for FaCT++ only 2.83%. A more detailed breakdown can be found in Table 6.9.

Not all variance is a consequence of experimental error. A varying subsumption test count is evidence for stochastic choices in the classification process. For example, a reasoner might require 10 tests to classify $\mathcal{O}$ in the first run, and 12 in

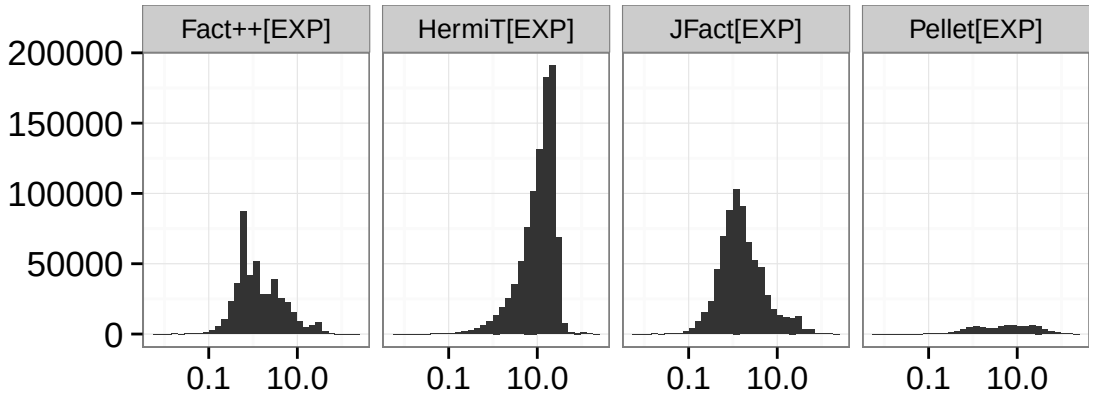


Figure 6.9: Histogram of COV of subsumption test measurements by reasoner (x:COV, log-scale).

Reasoner	Mean	Min	25%	Median	75%	Max
Fact++	2.83	0	0.52	0.96	2.93	167.87
HermiT	13.96	0	6.70	13.13	19.85	172.50
JFact	3.54	0	0.73	1.35	2.94	169.07
Pellet	13.24	0	1.60	5.96	18.27	172.65

Table 6.9: Variance of test measurements across reasoners (COV).

the second. With 242 out of 358 cases (67.6%) showing differences in the number of subsumption tests measured across runs, variation is high. 20 module-reasoner pairs (5.6%) vary by more than 10% in the number of subsumption tests. These results cannot be generalised to the entirety of BioPortal (given the nature of the sample), but they suggest that at least some of the measurement variance is not a consequence of measurement error, but of stochastic effects. Another indication of stochastic effects in the classification process is the number of times a particular test was triggered across a number of runs, for example twice across three runs. Any number other than the exact number of runs would be evidence for some stochastic effect. In practice, we found only 105 (less than 0.005 %) such tests that where *not* captured by the overall test count metric, i.e. that occurred in cases with varying overall test count.

Stochastic effects may not only be triggered by non-determinism in the tableau algorithm. FaCT++ for example does not reveal any evidence for stochastic effects during classification, while JFact does. This suggests that the source for stochastic effects might lie with data structures or methods in Java [GBE07] (as FaCT++ is implemented mainly in C++). Unfortunately, the source of the

stochasticity cannot be pinpointed by looking at test order alone: randomness in the implementation of the classification algorithm can induce changes in the exploration of non-deterministic branches of the tableau and the other way around. Effects of modularity may not be determined accurately in the presence of any strongly stochastic effects. In the following analysis, we will therefore often distinguish between those classifications for which we witnessed stochastic effects and those for which we did not.

Note that fluctuations in the number of tests or fluctuating (dis-)appearances of tests[10] across runs establishes merely a lower bound for the number of cases that are potentially subject to stochastic effects. There are other signs, such as variation in the test order, that could be signs of stochasticity. Unfortunately, neither test count, (dis-)appearance nor order, are by themselves explanatory of classification hardness. They can neither, as pointed out before, point to the source of the stochasticity, nor can they be used to quantify the degree of stochasticity, i.e. the ratio of non-deterministic and deterministic choices. Due to our three run policy, there is a risk that we are falsely attributing changes in individual test hardness to modularity (e.g. reduced test hardness) that was actually caused by stochasticity. For example, consider a test $ST(A, B, \mathcal{O})$ that is measured three times (across three runs) in the sub, and three times in the super-module, with all three measurements in the super-module being easier than all three measurements in the sub-module. It could be the case that the difference in signature (1) induced a test order that triggered randomly other tests *before* $ST(A, B, \mathcal{O})$ that do most of the work (cached pseudo-models or similar) or (2) randomly led the tableau-reasoner into harder non-deterministic branches. In this case it can be argued that the easyfication, if it is merely due to test order, is *not* due to modularity, and beneficial non-determinism can be forced merely by changing the test order. Assuming that the impact of stochastic effects is randomly distributed, observations made about the population (average, mean variation) should be stable. For the remaining chapter we call a test measurement *potentially subject to stochastic effects* (1) if the test was either triggered during a classification with varying test counts across runs or (2) if the test appears/disappears across runs of the same classification (but the overall number of tests remains the same).

[10] Again: we could have the same number of tests across runs, say 10, but in one run, $ST(A, B, \mathcal{O})$ is replaced by $ST(A, C, \mathcal{O})$.

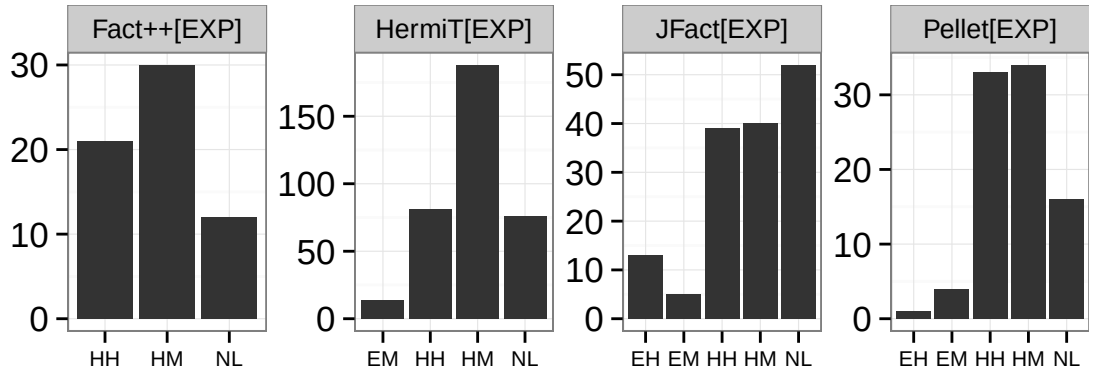


Figure 6.10: Hardness changes by reasoner: OCT. Bin labels x-axis: 1st letter: tendency (easier, neutral, harder), 2nd: magnitude (low, medium, high). Y-axis: number of comparisons.

What Effect does Modularity have on Subsumption Test Hardness?

In order to investigate the effect of modularity on subsumption test hardness we sampled 30 sub-module super-module pairs $\mathfrak{P}$ from the 120 possible combinations as described in Section 6.3.1. Discarding unsuccessful classifications, we obtained data from 703 out of 750 possible comparisons. For result stability, we excluded all pairs that (1) were incomplete, i.e. we measured for either the sub or the super-module less than 3 runs and (2) the sub and the super-module were equal. The following analysis is conducted on the remaining 659 cases (87.87% of 750).

Modules: There are 39 cases (around 5.7%) where the OCT of the sub-module is higher than that of the super-module, and 173 (25.1%) where there is no significant change in hardness (less than 5% change), compare also Figure 6.10.

From our 659 sub/super-module pair classifications, we obtained a total of 8,664,108 test comparisons. A comparison consists of

- a test $ST(A, B, \mathcal{O}, \mathcal{R})$,
- two modules $\mathcal{M}_{sub}, \mathcal{M}_{super}$ with $\mathcal{M}_{sub} \subset \mathcal{M}_{super} \subseteq \mathcal{O}$,
- a set of measurements $\{ST(A, B, \mathcal{M}_{sub}, \mathcal{R}, i) \mid 1 \subseteq i \subseteq 3\}$ and
- a set of measurements $\{ST(A, B, \mathcal{M}_{super}, \mathcal{R}, j) \mid 1 \subseteq j \subseteq 3\}$.

Sometimes, we refer to such comparisons between sub- and super-module as "cases". In what follows, we restrict our analysis to those cases where we have 3 measurements for each set. The reason for that is the following: During a single classification run, a test can be triggered for $CT(\mathcal{M}_{sub}, \mathcal{R})$ and not for $CT(\mathcal{M}_{sub}, \mathcal{R})$, or for $CT(\mathcal{M}_{super}, \mathcal{R})$ and not for $CT(\mathcal{M}_{super}, \mathcal{R})$. This fluctuating appearance is an indication that a test is subject to a stochastic effect. It is

unclear how do deal with tests that are occasionally absent. For example, they can either be ignored, or they can be counted as zero duration. Our interest lies solely in determining whether individual tests triggered during classification are harder or easier in the sub-module. Therefore, we restrict our treatment of potential stochastic effects to isolating ontologies with evidence for stochastic effects from those without, as described in the previous section.

Overall, subsumption tests are easier in the sub-module than in the super-module across all reasoners and module pairs in the set. Tests triggered in the super-module are *harder by a factor of 2* (mean). The difference between reasoners however is large: While for FaCT++ tests in the super-module are harder by a factor of 6.18, test hardness for JFact (0.42), Pellet (0.257) and HermiT (0.075) are much less affected. If we separate out ontologies with varying test counts as potential subjects to random effects, we find that HermiT finds tests *easier* in the super-module (-0.04 change), but harder when classifying ontologies with evidence for stochastic effects (0.08). Note that these changes are extremely low and cast doubt on the conjecture that HermiT could benefit from modularity in terms of subsumption test hardness reduction. It is quite possible that this low effect can be explained by architectural differences: the hyper-tableau implementation of HermiT mitigates the effect of algorithmic non-determinism for Horn ontologies by dealing with GCIs directly (rather then having to rewrite them into potentially costly, non-deterministic disjunctions) [GHM+14]. For Pellet we measured changes of 0.87 and 0.20 (potential stochastic effect) and for JFact 2.91 and -1.43 (potential stochastic effect). Table 6.10 presents an analysis of percentiles by reasoner. Note the potentially extreme effects of measurement error as a consequence of dealing with measurements in the microsecond area. The median, for all four reasoners, is around 0. Figure 6.11 confirms (note the log-scale on the y-axis) that the majority of tests are centred around 0, i.e. they do not change in hardness at all. Both JFact and FaCT++ however exhibit a bi-modal distribution, with at least one very distinct distribution of tests towards the hard end, i.e., tests that are significantly harder in the super- than in the sub-module. Interestingly, for almost all of these cases, the test in the sub-module differs from the test in the super-module by between 220 and 240 ms. This is quite a lot, given that the average difference between tests is around 0.56 milliseconds. Most likely, an expensive subsumption test in the super-module was replaced by a cheap look-up in the sub-module.

$\mathcal{R}$	Min	1%	5%	10%	25%	50%	75%	90%	95%	99%	Max
In cases with no evidence for stochastic effects in classification process.											
F	-556.60	-24.55	-1.17	-0.14	-0.02	0.00	0.02	0.15	2.28	200.73	2155.73
H	-3.17	-2.18	-0.53	-0.40	-0.19	0.00	0.12	0.36	0.54	1.00	2.33
J	-1099.59	-24.24	-0.81	-0.20	-0.03	0.00	0.03	0.11	0.28	150.54	2149.22
P	-15.90	-0.67	-0.17	-0.11	-0.02	0.05	0.62	1.66	3.67	14.79	136.26
In cases with evidence for stochastic effects in classification process.											
H	-264.02	-0.31	-0.18	-0.12	-0.03	0.05	0.16	0.28	0.36	0.53	278.37
J	-1083.87	-25.21	-19.18	-1.65	-0.08	-0.00	0.02	0.17	0.34	2.62	902.85
P	-287.54	-2.48	-0.72	-0.38	-0.09	-0.00	0.11	0.41	0.82	6.05	817.77

Table 6.10: Summary for the change of subsumption test hardness from sub- to super-module. Measure: redefined normalised fold change. 0 means no change, 0.5 means a change of +50%.

Given the low stability of many test measurements, we will present our analysis using our hardness change categories as introduced in Section 6.3.1, including tendency, magnitude, stability and potential stochastic effects. This will also clarify the separation of significant and insignificant changes. Figure 6.12 presents the full break down of the *pathological* cases with respect to our coding scheme (see Section 6.3.1). Almost 50% of the tests do not change in hardness significantly (optimal cases). The branch with the changes towards harder reflect our *expected* cases. The branch with the changes towards easier reflect our *pathological* cases. At least 2.82% of the tests are likely to be truly pathological: Significantly easier (more than 50%) in the super-module (therefore harder in the sub), and clear cut stability, i.e. cases where all three measurements in the sub-module are easier than all three measurements in the super-module.

Figure 6.13 presents the summary of the binning by hardness change category for all four reasoners. FaCT++ exhibits mostly neutral hardness changes, i.e. by far the majority of the tests triggered by FaCT++ do not change by more than 5%. This appears to contradict the observation that Pellet and JFact measurements vary a lot across almost all potential categories. The *pathological* cases as described in Section 6.1, corresponding to the EHC category and EHH (tests were much (H) easier (E) with high stability or even clear cut (H/C)), occur rarely (compare also the detailed breakdown in Figure 6.12). Out of the 1,653,732 tests that became easier overall, only 513,704 are of a high magnitude. The bins HHC and HHH correspond the category of *expected* behaviour. We can see in Figure 6.13 again that this behaviour is comparatively rare. *Optimal* behaviour is reflected in the NLH category and to a lesser extend by the NLL category, which

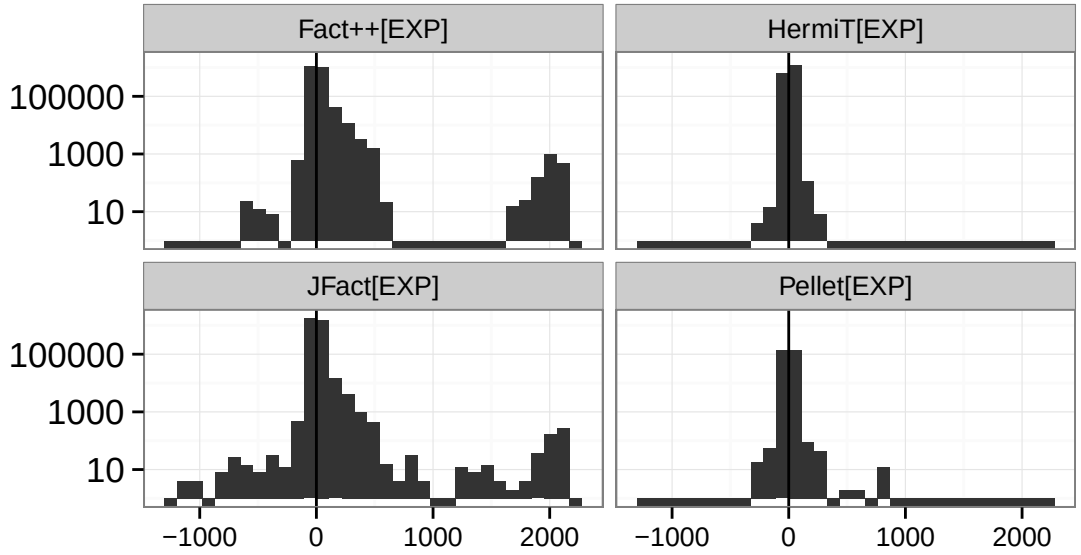


Figure 6.11: Histogram of change in hardness between sub and super-module. Counted are only such cases where the test was triggered in all three runs, both for the sub- and super-module.

involve hardness changes that are on average neutral but have a high standard deviation.[11] These categories are the *by far* dominant category, compare also the neutral (N) changes in Figure 6.12 (52.65%). Interestingly, there appears to be no correlation at all between the difference in the size of the super- and sub-module to the change in hardness (Pearson correlation coefficient less than 0.01).

Discussion On average, reasoners appear to profit from modularity in terms of subsumption test hardness, i.e. they are able to ignore irrelevant axioms, therefore easifying tests. The magnitude however is, depending on the reasoner, surprisingly low. For Pellet and HermiT at least, pathological changes in subsumption test hardness are often cancelled out by expected positive changes in hardness. This observation suggests that picking a single subsumption test and tracing its hardness through sub- and super-modules in isolation may be misleading. The majority of tests do not change at all under modularity (more than 50%). The positive results for FaCT++ and (to a lesser extend) JFact (easyfication by a factor of 6.18 and 0.42 respectively) are due to a comparatively small number of tests that have extreme changes in hardness, which might be indications for bugs,

[11]The fact that HermiT alone has mostly unstable tests among the neutral cases cannot (currently) be explained.

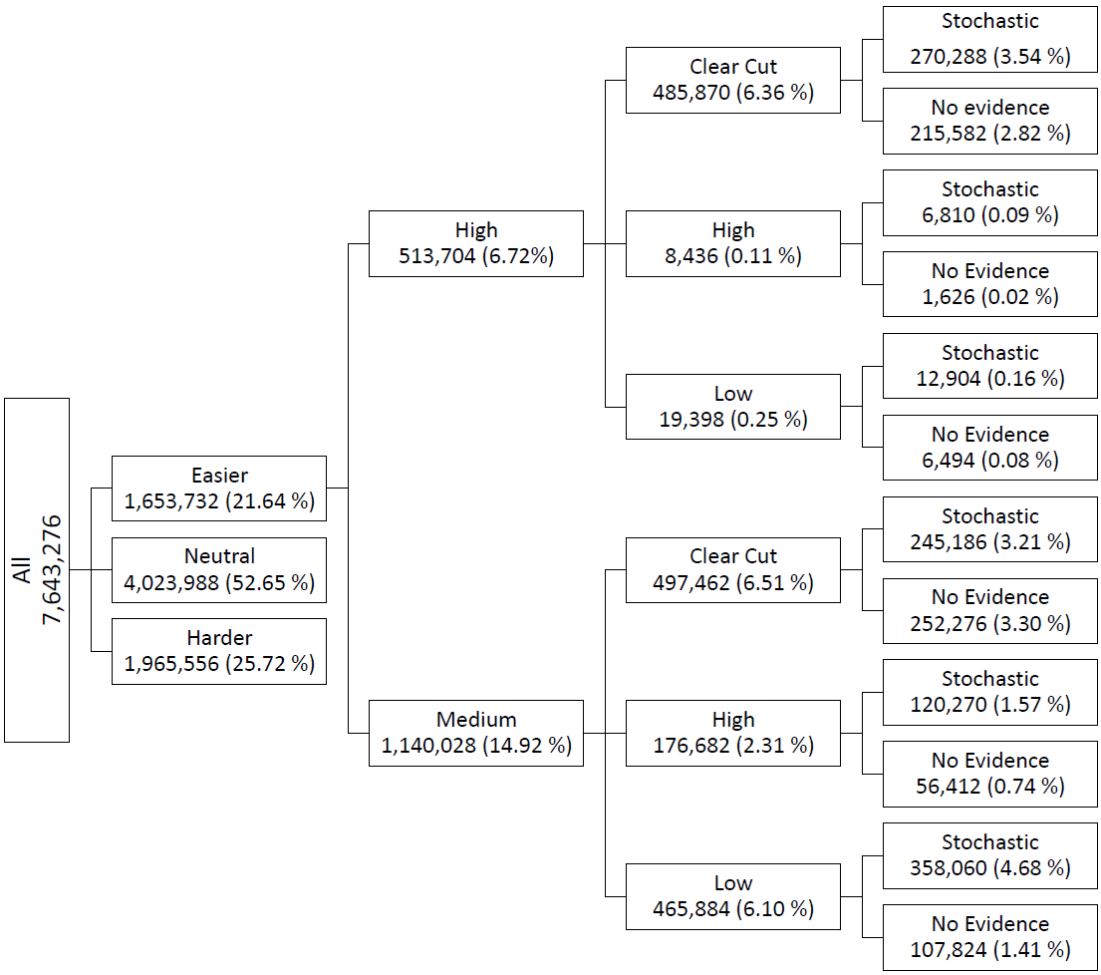


Figure 6.12: Breakdown of hardness changes for the subsumption tests in the data set that were easier in the super-module (first branch). The second branching reflects the magnitude of the change, the third the stability of the measurement. The number of tests in the "stochastic" category reflects the number of tests for which there was some evidence that the entire classification process (the test was part of) was subject to stochastic effects. Percentages are with respect to overall number of cases.

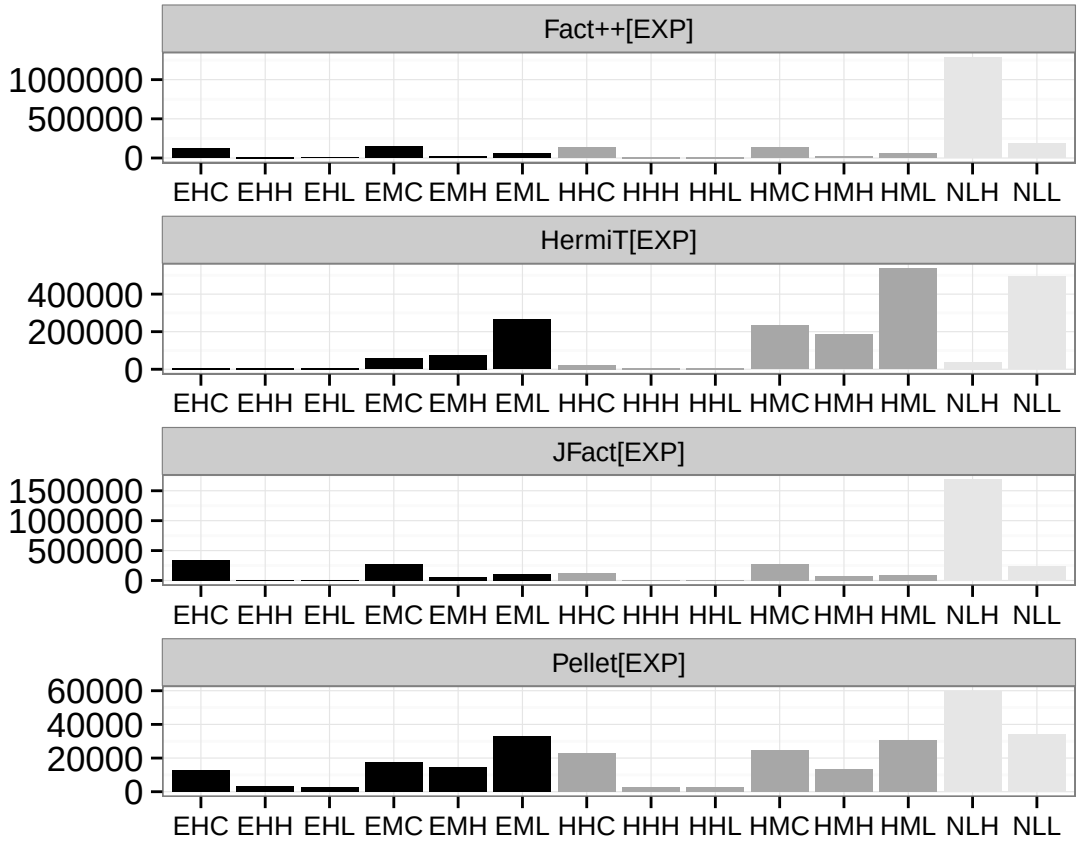


Figure 6.13: Hardness changes by reasoner: SST. Bin labels x-axis: 1st letter: tendency (easier, neutral, harder), 2nd: magnitude (low, medium, high), 3rd: stability: (clearcut, high, low). Y-axis: number of comparisons. The EHC and EHH categories correspond to our *pathological cases*, the HHC and HHH cases to the *expected cases* and the NLH and NLL to the category of *optimal cases*. The remaining categories are, due to the low number of runs and high variation, neither clearly pathological nor expected.

for example during absorption. It remains unclear whether the small number of significantly pathological tests (2.82%), are really pathological, i.e. the consequence of a bug in the implementation, or whether their hardness merely shifted to another test. This can be triggered for example by a changed traversal order due to the difference in signature between the two modules, or random effects in the classification process that we have not isolated. The current experimental setup merely attempts to determine a tendency, i.e. *whether* reasoners are sensitive to axioms that are irrelevant to a particular set of entailments (Research Question 6.2).

Methodological Reflection

Our original goal was to be able to determine the hardness change of a particular subsumption test $ST(A, B, \mathcal{O}, \mathcal{R})$ from a sub- to a super-module. We learned that some tests get significantly harder, most tests do not change in hardness, but some tests also get significantly easier. We therefore conclude that it is insufficient to trace a particular set of tests - the whole population of tests have to be studied at once. We believe that the current experimental design allows for identifying a tendency, but any conclusions about the magnitude of the overall effect should be avoided. First, we ignore tests that are triggered in only one of the two modules entirely. It is possible that the changes in hardness shift to tests outside of the range of tests we are observing. Second, we cannot conclusively isolate stochastic effects if we do not observe the internals of any given subsumption test, and consider changing test orders. It remains to be seen what causes the changes in hardness exactly. Lastly, the sample of ontologies was not meant to be representative of the population. A representative sample would take by far too long to be processed with the current analytic pipeline.

As a side observation, we noted the high average variation between individual test measurements across runs. This is most likely a consequence both of stochastic effects and measurement error, due to the lack of accuracy of our timing methods (see Section 4.4.2). For any experiments that cannot be statistically significant because of time constraints, this means that the threshold for effects related to individual subsumption tests should be set higher than 5% (we used either 50%, or clear cut).

In this chapter, we have focused on subsumption test hardness and deliberately omitted looking at subsumption test avoidance and traversal. In the next

chapter, we will study what difference subsumption test hardness *and* avoidance makes for decomposition-based classification.

6.5 Summary of Key Observations

In the following, we will briefly summarize the key observations made in this chapter:

- We re-confirm the almost 20 years old results by Horrocks [Hor97] that subsumption tests are generally rather easy.
- The impact of tableau subsumption testing is significant only for a *small number of ontologies*, which threatens the applicability of modular techniques to reduce the number of tests or test hardness, at least for ontologies within the performance range of this survey. No conclusions can be drawn on the applicability of modular techniques for ontologies (1) outside these bounds (i.e., $CT(\mathcal{O}, \mathcal{M}) \geq 1hr$) and (2) for a reason other than subsumption test hardness reduction or avoidance (e.g. parallelism, efficient delegate).
- Two thirds of the ontology-reasoner pairs in our sample show strong evidence for random effects in the classification process (varying test counts across runs and absence and presence of individual tests). This, and the low number of feasible runs, preclude observation of changes in hardness for individual subsumption tests (rather than for the entire population).
- Subsumption test hardness increased on average by a factor of 2 between the sub- and super-module pairs sampled as part of Experiment 6.2.
- However, the majority (more than 50%) of the tests in our sample did not change at all or only to a very small degree in hardness between the sub- and the super-module.
- We have isolated pathological cases of tests being easier in a super than in a sub-module from obvious stochastic effects. However, easier tests are often accompanied by harder ones, which suggests that often, the hardness merely *shifts* from one test to another.
- HermiT and Pellet appear to be sensitive to modularly irrelevant axioms only to some small degree. For HermiT, this may be due to mitigated non-determinism (see Section 6.4.2). This limits the potential of modules to reduce subumption test hardness. JFact and FaCT++ have greater

reductions in test hardness on average under modularity. The experimental design was sufficient only for exploratory analysis: Larger studies are needed to determine whether a particular reasoner would *generally* profit from the protective effect of modularity.

Chapter 7

Modular Classification in Action

In the previous chapters, we have established the risk of pathological modules for modularity-based techniques for Description Logic classification (Chapter 5) and analysed the effect of modularity on subsumption test hardness. In this chapter we will explore how modular techniques fare in practice, measuring their overhead (see Section 3.5.1), and potential benefits for subsumption test avoidance and hardness reduction.

7.1 Definitions and Models

The phenomenon under investigation is the viability of modular and module-based decompositions (see Section 2.5) for improving Description Logic classification performance. We quantify performance as execution time, operationalised as described in Section 4.4.2.

The intuition behind the assumption that modularity-based classification might be superior compared to traditional monolithic classification has been discussed extensively in Chapter 3. Given an ontology $\mathcal{O}$, a reasoner $\mathcal{R}$, a module $\mathcal{M}$ of $\mathcal{O}$ with $\mathcal{M} \subseteq \mathcal{O}$, two concept names $A, B \in \widetilde{\mathcal{M}}$ (and therefore in $\widetilde{\mathcal{O}}$), $\mathcal{T}_{\mathcal{R}}^{\mathcal{O}}$ being the traversal space (Definition 2.1) of reasoner $\mathcal{R}$ during the classification of $\mathcal{O}$ and $ST(A, B, \mathcal{O}, \mathcal{R})$ a subsumption test triggered by $\mathcal{R}$ during the classification of $\mathcal{O}$; we conjecture that:

- $\mathsf{time}(ST(A, B, \mathcal{O}, \mathcal{R})) \geq \mathsf{time}(ST(A, B, \mathcal{M}, \mathcal{R}))$ (see Section 3.4.1)
- $|\mathcal{T}_{\mathcal{R}}^{\mathcal{O}}| \geq |\mathcal{T}_{\mathcal{R}}^{\mathcal{M}}|$ (see Section 3.4.2)

We have described our model of modular reasoning in Section 3.2. In order to assess the performance of modular techniques, we mainly rely on comparisons to the monolithic approach. While our default model of decomposition-based classification is a pure blackbox model, i.e. we assume that each chunk is processed individually and there is no communication between the delegates, we want to investigate a number of what-if scenarios or analytic models, described in the

following.

The second scenario or model—the first being the blackbox model—is the *encoded decomposition*. We know from experience that computing the decomposition constitutes a major overhead for modular techniques. In Section 3.5.1 we have discussed the possibility of encoding the modular structure into the serialisations of the ontology. Note that we take a simplified view on decoding the modular structure and assume that it does not add any overhead to the de-serialisation of the ontology.

The third model is the *glassbox model*. One of the sources of overhead of decomposition-based classification is redundancy, a consequence of overlapping chunks in the decomposition. In order to avoid re-doing work, for example determining a particular subsumption or non-subsumption more than once, we need to communicate results between delegate reasoners. Our default scenario does not involve any communication between the different delegate reasoners. The original MORe approach involves a minimum amount of communication by adding already determined subsumptions from the classification of the DL (remainder) module to the $\mathcal{L}$-module before ELK processes it (see Section 3.3.1). That way, the result of the $\mathcal{L}$-module classification equals the classification $\mathcal{O}$. However, consequence-based reasoners such as ELK do not need to communicate known non-subsumptions. For traversal/ (hyper-) tableau style delegate reasoners, we model the glassbox approach in the following way. We assume, perhaps naively, that delegate reasoners have access to a shared representation of known subsumptions—and non-subsumptions. That way, once the a delegate reasoner has determined that $\mathcal{O} \models A \sqsubseteq B$ or $\mathcal{O} \not\models A \sqsubseteq B$, no subsequent delegate reasoner with A and B in its signature will have to trigger that test.

The fourth model is simply a combination of the second and the third model, i.e. the encoded chunking and glassbox model combined. While this scenario will not give us many additional insights into the performance of modular techniques, it will give us a better sense of the potential benefit of decomposition-based classification. However, since our glass box model is naive (there are a number of sources of redundancy unaccounted for, like redundancy during pre-processing), it does not reflect the maximum potential of decomposition-based classification.

7.2 Implemented Modular Classification Strategies

In this section, we will describe the five modular classification approaches we have considered in our experiment, two of which reflect the known approaches involving modular decompositions as implemented by MORe and Chainsaw and three (novel) experimental approaches based on the Atomic Decomposition (modularity-based decompositions). We have discussed the differences between modular and module-based decompositions in Section 2.5.

Note that we measure the monolithic approach in the same way as we measure the modular approaches: as part of Katana. The monolithic approach therefore goes through the same stages as any modular reasoner would, with the difference that the decomposition phase is omitted, the dispatch is simply an assignment of the whole ontology to the only delegate reasoner, and the classification stage involves only a single delegation.

All five approaches/strategies are listed in the following:

- The *Maximal Module* approach is based on the Atomic Decomposition and classifies the maximal modules in the Atomic Decomposition one-by-one (MM). This approach is similar, but not the same, as the one employed by Chainsaw.
- The *MORe* approach (Section 3.3.1) is based on the MORe decomposition of $\mathcal{O}$ into an $\mathcal{L}$-module and a remainder module (MOR).
- The *Connected Component* approach is based on the Atomic Decomposition and involves classifying the set of connected components in the Atomic Decomposition one-by-one (CC).
- The *Optimised Connected Component* approach is based on the CC approach, summarising small connected components into larger ones to mitigate the overhead of creating overly many delegations (CCO).
- The Community Detection-based approach is based on the Chainsaw approach, summarising sets of strongly connected maximal modules (communities) to chunks of similar sizes in order to mitigate the overhead of creating overly many delegations (CD).

All strategies share two core components: (1) A single factory for (a place to instantiate) full-fledged OWL 2 delegate reasoners and a second factory for OWL 2 EL delegate reasoners. (2) An optional mechanism to dispatch a subset

to an optimised $\mathcal{EL}$-delegate reasoner (a flag that, if set, checks the profile of a chunk and assigns it to an optimised delegate).[1]

7.2.1 Chainsaw Strategy (MM)

The Chainsaw approach is based on the Atomic Decomposition and works essentially on the assumption that, if the ontology is decomposed into very small modules, the subsequent pruning of the search space (and possibly the reduction of the test hardness) outweighs the overhead induced by computing the decomposition. All maximal modules (or root modules, see Section 2.5.1) of the Atomic Decomposition are classified as they come, with the results being, at each classification, merged into a unified class hierarchy. The approach was further described in Section 3.3.2.

The main *threat* from employing the MM strategy[2] in a black-box fashion is the potentially huge redundancy induced by the overlap of its chunks. Note that Chainsaw is doing a bit more: It avoids subsumption tests based on certain properties of the labelled Atomic Decomposition. As current (delegate) OWL reasoners perform full classification before answering individual subsumption queries, this additional traversal layer does not make any practical difference, see Section 3.3.2. However, the Katana framework could be easily extended to accommodate, for example, for an Atomic Decomposition-sensitive traversal.

As all chunks in the decomposition correspond to modules, we say that the MM decomposition is a modular decomposition (Definition 2.5).

7.2.2 MORe Strategy (MOR and MORA)

The MORe-approach was described in detail in Section 3.3.1. Note that we strictly distinguish between the MORe-decomposition, which is formed by computing two (potentially overlapping) modules according to a particular method and the MORe-classification strategy, which is (at least approximately) defined as classifying the OWL module first with a full fledged OWL reasoner and then computing the remaining class hierarchy by classifying the $\mathcal{L}$-module with ELK (Section 3.3.1). Note that in Katana (and in this Chapter), we consider the

[1] The dynamic dispatch feature is implemented, but will not be used as part of the experiments in this Chapter.

[2] Apart from the potentially severe cost of computing the decomposition, a general risk shared by all modular strategies, most importantly by those building on the Atomic Decomposition.

MORe $\mathcal{L}$-module to be a module for ELK. In this chapter we also investigate a slight modification of MOR, MORA, which simply dispatches both modules in the decomposition to the full fledged OWL reasoner delegate.

The *main threats* to the MORe approach are considerable overlap of the two modules, and that the DL (remainder) module is so large that classification can suffer from the unfriendly non-determinism observed in Chapter 5.

Like the MM decomposition, the MORe-decomposition is classified as modular, as both chunks that make up the decomposition are modules.

7.2.3 Connected Components Strategy (CC)

The Connected Component strategy is based on the experience that the redundancy induced by the overlap of the set of maximal modules is critically harming performance (MM, or Chainsaw, strategy). Connected components are logically independent from each other (see Section 2.5.1). It is important to understand that a single connected component can be, but is not necessarily, a module. The first obvious case where a connected component corresponds to a module is if it contains only a single maximal module (i.e., the connected component would simply be equal to that maximal module). If the connected component however contains more than one maximal module, it does not necessarily correspond to a module.[3] Another (obvious) case where the connected component corresponds to a module is if it is equivalent to the entire ontology. Following the CC strategy, the reasoner first computes the Atomic Decomposition, and then delegates each connected component in the decomposition to a separate delegate reasoner.

The most important *threat* to the connected component strategy is the *large main–tiny satellites* effect: There may be a large number of connected components, but one of them is huge (or in the worst case corresponds to the whole), and the others are tiny, in practice sometimes only a single axiom. This may be harmful because (1) the main component may be almost as big as the parent ontology, thus voiding almost all potential gains from modularity and (2) a potentially large number of tiny connected components requires a large amount of delegate reasoner instantiations. Another threat emanates from the fact that connected components may lose the protectiveness of modularity, which may cause them, in the worst case scenario, to behave like hard subsets, i.e., they may be harder to classify than the whole ontology as discussed in Chapter 5.

[3] See page 100 in PhD thesis of Del Vescovo for an example [DV13].

As connected components may or may not correspond to modules, and each connected component can be described as a union of its maximal modules, we classify the strategy as classification-preserving (module-based), but not modular (chunks do not necessarily correspond to modules).

7.2.4 Optimised Connected Component Strategy (CCO)

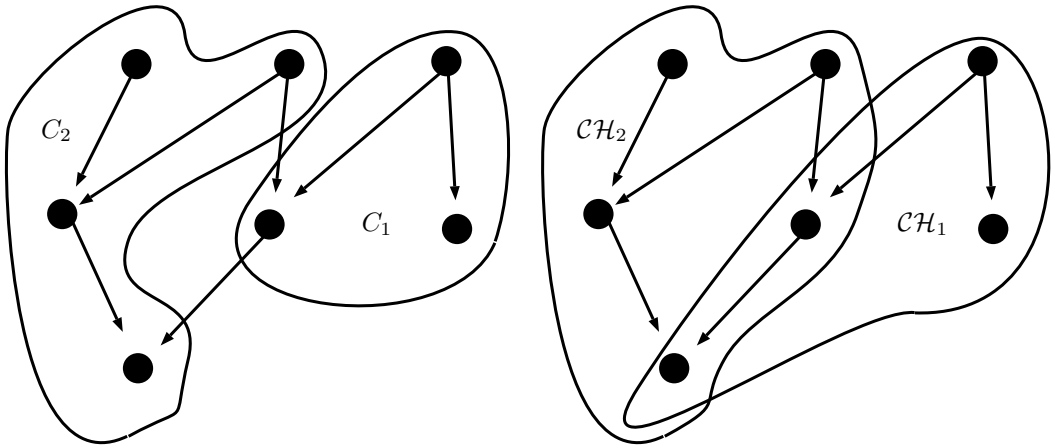
A straightforward answer to deal with the large number of delegations to trivial chunks of the large main—tiny satellites problem is to merge small components together. This "collapsing" is implemented in a slightly naive fashion as follows. We first determine the size of the largest chunk. Then we define a quarter of the size of the largest chunk as the maximum threshold for a merged chunk. Following that, we iterate through the set of connected components. If a component is smaller than the threshold, we find the smallest chunk smaller than the threshold and add the component to it (i.e. merge them). This is done until there is at most one chunk left that is smaller than the threshold. We call this strategy Optimised Connected Component strategy. Choosing a quarter of the largest chunk as a threshold means that new chunks as a result of a merge can never be more than half the size of the largest component.

While this does not address the problem of a single very large connected component, it mitigates the overhead induced by creating a large number of delegate reasoners for potentially tiny, sometimes single axiom, connected components. With the same rationale as the CC strategy, the underlying decomposition of this approach is classified as classification-preserving, but not modular.

7.2.5 Atomic Decomposition Community Detection Strategy (CD)

The main threat of the MM approach is that the maximal modules in the decomposition overlap to such an extend that potential benefits of search space pruning are outweighed by the redundancy induced by the overlap. On the other hand, the CC strategy might have little to no redundancy (logically disconnected chunks), but suffers from the large main–tiny satellites problem. A compromise between the two is to summarize somehow stronger connected sets of maximal modules into chunks.

A known approach in graph theory to partition a graph into sub-graphs that



(a) Communities. (b) Chunks.

Figure 7.1: An example of the community detection-based method. Nodes and arrows represent the atoms and dependencies of the Atomic Decomposition. Left: communities (C_1 and C_2) as determined by the community detection algorithm. Right: corresponding chunks ($\mathcal{CH}_1$, $\mathcal{CH}_2$) as extracted from the set of communities.

are densely connected inside, but more sparsely connected outside is community detection [For10]. As the Atomic Decomposition is a graph, we generate chunks in the following way. We first partition Atomic Decomposition as a set of communities using the implementation of the modularity based community detection in Gephi [BHJ09], based on an algorithm by Blondel et al. [BGLL08]. The set of communities all by themselves is not classification-complete (see Section 2.5) because, for any given atom part of some community, it is not guaranteed that all its dependencies are also in the same community. Therefore, we re-define a chunk as the union of maximal modules corresponding to the set of maximal atoms in a community. That way, not only is it possible that axioms in the original community are discarded—if a community does not include any top atoms, it is discarded entirely. In other words, this approach attempts to summarise the set of maximal modules to larger chunks that are more strongly connected internally, that way (hopefully) merging some of the maximal modules with very large overlap and reducing the number of necessary delegate reasoners. As each vertex in a graph belongs to exactly one community, no maximal atom is present in more than one chunk. In order to mitigate the risk of potentially many chunks, we employ the same method as the CCO strategy to merge very small chunks.

The *main threat* to this approach is the overhead of computing the communities (on top of the danger that the redundancy is still outweighing the gains in terms of traversal space pruning). The decomposition produced by the CD approach is classified as classification-preserving, but not modular.

Table 7.1 shows a summary of the approaches with threats and rationale.

Table 7.1: Overview of modular and module-based decompositions analysed in this chapter

	$D_{\mathcal{O}}$	Chunking	Rationale	Primary Threat
MORe	EL + OWL	Decomposition	Cheap EL classification, small remainder	Pathological remainder, overlap
MM	AD	Set of all maximal modules in AD	Smallest possible chunks	Large overlap
CC	AD	CC's in AD	Mitigate redundancy of MM	large main– tiny satellites
CCO	AD	CC's in AD, small ones merged	Mitigates CC overhead caused by small chunks	Large main
CD	AD	Communities in AD, small ones merged	Possibly smaller stronger related chunks than CC	Large overlap, computing communities

7.3 Empirical Characterisation

In this chapter, we are addressing the following question:

Research Question 7.1. *Are decomposition-based reasoning techniques suitable to improve classification performance?*

In order to address this question, we will analyse the five modular classification approaches described in Section 7.2, specifically with respect to the following **five dimensions**:

- Overall performance
- Potential for reasoning hardness reduction in general and test hardness reduction in particular (Section 3.4.1),
- Potential for test avoidance (Section 3.4.2),

- Threat of subsumption testing redundancy (Section 3.5.3) and
- Threat of overhead (Section 3.5.1).

Because of their close connection, we will analyse the potential for test avoidance and the threat of test redundancy as well as the overall performance and computational overhead together. For a reminder of some of the measures (fold change, coefficient of variation) used, please see Section 4.5.

7.3.1 Overall Performance

In order to determine the performance of all five (six, if we count MORA) decomposition-based classification approaches, we pitch them against the monolithic version of their primary delegate reasoner. There are two key questions which will be analysed with respect to overall performance: (1) Which methods show the most promise? (2) How do factors of interest contribute performance? We will describe the metrics of comparison and the operationalisation of our models of modular approaches (see Section 7.1) in the following.

Table 7.2 shows an overview of how we quantify the performances of each technique according to our (four) models. The main metric of performance is the *overall classification time* (OCT): The duration of the entire reasoning time (including pre- and post-processing).

Considering the *encoded chunking scenario*, the OCT will be re-computed by simply reducing it by the time it takes to compute the decomposition. For the *glassbox model*, we re-calculate the overall classification time by subtracting the sum of redundant test times and the sum of initial consistency check test times. The set of redundant tests is defined as follows:

Definition 7.1. *Given a reasoner $\mathcal{R}$, an ontology $\mathcal{O}$, I a classification run of $\mathcal{O}$ by $\mathcal{R}$, A and B two names with $A, B \in \widetilde{\mathcal{O}}$, $ST(A, B, \mathcal{O}, \mathcal{R}, I)$ a single subsumption test triggered during I to determine whether $\mathcal{O} \models A \sqsubseteq B$ and $\mathfrak{S}_{\mathfrak{all}}$ the set of all subsumption tests triggered during I, then we call:*

- $\mathfrak{S}_{un}$ *the set of* unique *subsumption tests if for each $ST(A, B, \mathcal{O}, \mathcal{R}, I) \in \mathfrak{S}_{un}$ there is* no easier test $ST(A, B, \mathcal{O}, \mathcal{R}, I) \in \mathfrak{S}_{all}$ *and*
- $\mathfrak{S}_{\mathfrak{all}} \setminus \mathfrak{S}_{\mathfrak{un}}$ *the set of* redundant *subsumption tests.*

For each of the classification approaches we want to find out (1) are they perform better compared to the monolithic case and (2) how do they fare against

the other decomposition-based classification approaches. We defined a scoring system for the performance of a technique as follows:

Procedure 7.1. *Scoring classification performance*

1. *Select a metric of interest.*
2. *For each ontology and primary delegate, determine a ranking of the modular classification strategies based on the OCT (distinguishing between all four scenarios; blackbox, glassbox, encoded chunking, glassbox + encoded chunking), with (rank) 1 being the best.*
3. *From the ranking, assign 5 points to a technique whenever it ranked first (for any ontology, primary delegate and model), 4 points if it ranked second, 3 if third, 2 if fourth, 1 if fifth and 0 points if it ranked lower less than fifth or failed to produce the metric (i.e. to classify in time) altogether.*
4. *The final score for each technique (distinguishing between activated $\mathcal{EL}$-Dispatch, primary delegate and classification model) is computed by adding all points together across ontologies.*

Metric: Overall Classification Time

In the following, we present the (performance-wise) dominant sub-processes contributing to the overall classification time (OCT), all of which will be taken into account when analysing the OCT. For an overview of the entire process, refer back to Figure 3.2.

- Preprocessing time: We will, for all classification strategies in an equal manner, ignore the initial consistency check during M1 (which is totally redundant, given that consistency is checked for each chunk). That means that preprocessing involves decomposition, (possibly) chunking and delegation.
  - Decomposition time: Computing the modular decomposition is, especially in cases that involve the Atomic Decomposition, sometimes the biggest contributor to OCT. Therefore, we report the decomposition time separately from the (rest of the) preprocessing time.
- Modular Classification: the time it takes to compute the partial class hierarchies for every chunk and delegate reasoner. The two main contributors of interest to this stage are:
  - Sum of initial consistency check times (SCC): This number is of interest, because (1) it is often a primary contributor to OCT and (2) it is

clearly avoidable (once we establish the consistency of a single module, we know it for the rest).

- Sum of subsumption test times (SST): This metric is a key metric to assess the performance of modular classification, and represents the overall consequence of test avoidance, test redundancy and changes in hardness. Depending on the scenario (blackbox, etc.), this metric includes redundant tests. In the glassbox scenario, this metric is reduced to the sum of unique tests (Definition 7.1).
- Hardest delegation (HDL): The classification time of the hardest chunk.

All metrics will be reported with respect to their contribution to the overall classification time (OCT), i.e $\frac{METRIC}{OCT}$.

7.3.2 Traversal Space

The second dimension we analyse for all five classification approaches is their potential for *avoiding tests* and their proneness to *redundancy*. Again, this will be done by comparing each approach to the monolithic case represented by the behaviour of the primary delegate reasoner of the approach. For the comparison against the monolithic case, we need two metrics: Avoided tests and extra tests, defined as follows:

Definition 7.2. *Given a monolithic reasoner $\mathcal{R}_{mon}$, a modular reasoner $\mathcal{R}_{mod}$, an ontology $\mathcal{O}$, i a classification run of $\mathcal{O}$ by $\mathcal{R}_{mon}$, j a classification run of $\mathcal{O}$ by $\mathcal{R}_{mod}$, A and B two names with $A, B \in \widetilde{\mathcal{O}}$, $ST(A, B, \mathcal{O}, \mathcal{R}, i)$ a single subsumption test triggered during i to determine whether $\mathcal{O} \models A \sqsubseteq B$, $\mathfrak{S}_{\mathfrak{mon}}$ the set of all subsumption tests triggered during i and $\mathfrak{S}_{\mathfrak{mod}}$ the set of all subsumption tests triggered during j, then we call $ST(A, B, \mathcal{O}, \mathcal{R}_{mod}, j)$:*

- avoided *if* $ST(A, B, \mathcal{O}, \mathcal{R}_{mon}, i) \in \mathfrak{S}_{mon}$ *and* $ST(A, B, \mathcal{O}, \mathcal{R}_{mod}, j) \notin \mathfrak{S}_{mod}$,
- common *if* $ST(A, B, \mathcal{O}, \mathcal{R}_{mon}, i) \in \mathfrak{S}_{mon}$ *and* $ST(A, B, \mathcal{O}, \mathcal{R}_{mod}, j) \in \mathfrak{S}_{mod}$ *and*
- extra *if* $ST(A, B, \mathcal{O}, \mathcal{R}_{mon}, i) \notin \mathfrak{S}_{mon}$ *and* $ST(A, B, \mathcal{O}, \mathcal{R}_{mod}, j) \in \mathfrak{S}_{mod}$.

In other words, avoided tests are those triggered during the classification by the monolithic reasoner that were *not* triggered during the classification of the modularity-based approach. Extra tests are those tests that were triggered during the modularity based approach (potentially multiple times!) that were not triggered by the monolithic reasoner. Common tests were triggered by both.

We will quantify aspects of the traversal space using two groups of metrics (1) Attributes of the classification approach (redundancy) and (2) attributes of the comparison to the *monolithic case* (avoided and extra tests) in the following:

- Attributes of the classification approach
  - Total test count (STC): The total number of tests triggered during a modular classification.
  - Unique test count (UTC): The number of *unique tests* triggered.
  - Unique test time (UTT): The sum of all unique tests triggered.
  - Redundant test count (RTC): STC-UTC
  - Redundant test times (RTT): SST-UTT
- Attributes of the comparison to the monolithic case
  - Avoided test count (ATC): The number of *unique tests* triggered by the monolithic reasoner *not* triggered by the modular approach.
  - Avoided test total time (ATT): The sum of all avoided test times.
  - Extra test count *compared to the monolithic case* (ETC): The number of tests triggered by the modular approach *not* triggered by the monolithic approach.
  - Extra test times *compared to the monolithic case* (ETT): The sum of test times triggered by the modular approach *not* triggered by the monolithic approach.
  - Common test count modular reasoner (CTC): The number of tests triggered by both the monolithic and the modular approach
  - Common test times modular reasoner (CTT-MOD): the sum of test times of all common tests as the modular approach triggered them.
  - Common test times monolithic reasoner (CTT-MON): the sum of test times of all common tests as the monolithic approach triggered them.

We further produce an avoidance score, which is computed as follows (always from the perspective of the modular reasoner): We first determine the ratio of ATC-ETC to the common test count +1 (CTC) ($\frac{ATC-ETC}{CTC+1}$); the +1 one is needed to accommodate for the case that CTC equals 0. If the resulting value is positive, then we say that the modular technique at hand has a tendency of *avoiding* tests. If it is negative, we say that the modular technique has a tendency of adding *extra* tests. If ATC=ETC, we say that avoided and extra tests *cancel each other out*. We further distinguish between four levels of magnitude. If the number of avoided/extra tests is more than 10 times the number of common tests, we say

the effect is of *high* magnitude. Otherwise, if the number is more than equal to the number of common tests, we say the effect is of *medium* magnitude. If the number is more than 10% of the number of CTC, the magnitude is *low*, else it is *neutral*. This computation of the avoidance score relates to the blackbox model. When considering the glassbox model, ETC is replaced by unique extra test count, and CTC by unique common test count.

The effect of modularity on subsumption test hardness has been investigated extensively in Chapter 6. The only additional aspect we will investigate as part of this chapter is how beneficial/harmful modular reasoning techniques are compared to each other and compared to the monolithic case. Our primary metrics to quantify the effect of a classification approach on test hardness is mean subsumption test hardness.

All dimensions of the analysis are summarised in Table 7.2.

Table 7.2: Summary of metrics used to quantify key aspects of dimensions, by classification model.

Classific. Model	Dimension of analysis	Central metrics for analysis
BB	Performance	OCT
	Traversal	Impact analysis of avoided, extra and redundant tests
	Test Hardness	Descriptive statistics of test times
EC	Performance	OCT, w/o decomposition
	Traversal	-
	Test Hardness	-
GB	Performance	OCT, w/o redundant tests, w/o delegate consistency
	Traversal	Impact analysis of avoided and extra tests
	Test Hardness	Descriptive statistics of test times
EC+GB	Performance	OCT, w/o redundant tests, w/o delegate consistency, w/o decomposition
	Traversal	-
	Test Hardness	-

7.4 Experimental Design

We have conducted our experiment using three of the four (modified) reasoners described in Chapter 6, HermiT, Pellet and JFact, embedded into the Katana Framework, as extensively described in Section 4.2:[4]

Experiment 7.1. *Benchmark of 5 decomposition-based classification approaches, involving the analysis of overall performance, overhead and subsumption test hardness, avoidance and redundancy.*

We ran the experiment on two corpora: The first corpus was based on the *19 Pure DL* BioPortal ontologies that involved all those ontologies we processed as sampled for Experiment 6.2, *for which at least one reasoner triggered a subsumption test that was harder than 100 ms* (corpus henceforth called K1). Three ontologies were excluded from the analysis, two out of which caused out-of-memory exceptions during the analysis (NEMO, NCIT), and one caused at least one delegate reasoner to trigger the same test multiple times (OBI).[5] All experiments were randomly distributed over machine cluster 4.1, with a timeout of 2 hours each,[6] and repeated 3 times. The second corpus was based on the 39 ontologies analysed as part of Experiment 6.1 (henceforth called K2) which caused all four reasoners to trigger at least one subsumption test during classification. Four ontologies were excluded: for the same reason as before, NEMO and NCIT, and CPRO and CSEO again because of duplicate subsumption tests. The experiment was, due to its considerable runtime of more than two machine months, run on 8 instances of Amazons EC2 (machine cluster 4.2; experiments randomly distributed across all instances). Due to the much higher completion rates (next section) of K2, this will be our primary corpus of analysis. Interesting differences with K1 will be reported where appropriate.

[4]The experiment has a runtime of around a month. As we are primarily interested in the modular classification strategies, we decided to restrict ourselves to only three different delegate reasoners. Since FaCT++ used a slightly different method (C-based) for making time measurements (making the analysis a bit more complicated), we decided to focus on the delegate reasoners implemented in Java.

[5]The bug has been reported to the developers of FaCT++ and JFact, and was reported fixed for the latest release.

[6]Due to the potentially high cost of classifying many modules, we give a more generous timeout than in the previous experiments. The necessity for the timeout due to resource limitations remains the same.

7.4.1 Experimental Pipeline

For the experiment, we ran Katana with the following 7 configurations:

1. MOR: MORe decomposition with MORe strategy
2. MORA: MORe decomposition with default strategy (using full OWL 2 reasoner on both modules)
3. MM: Atomic Decomposition with maximal module strategy
4. CC: Atomic Decomposition with connected component strategy
5. CD: Atomic Decomposition with community detection based strategy
6. CCO: Atomic Decomposition with optimised connected component strategy
7. MON: no decomposition, default strategy

Item 7 reflects the "normal" monolithic strategy. Details on Katana can be found in Section 4.2.

7.5 Results

Out of the 1,008 classification runs (16 ontologies, 3 reasoners, 7 configurations, 3 runs) over corpus K1, 644 (63.89 %) successfully terminated, i.e. produced a classification. Out of the 2,250 classification runs (35 ontologies, 3 reasoners, 7 configurations, 3 runs) over corpus K2, 2,170 (98.41%) successfully terminated. For a full break down of completition rates by technique, see Table 7.3. The huge difference in completion rates can be explained by the difference in hardness: On average, ontologies in K2 took 24.9 seconds to classify, and ontologies in K1 479.8 seconds (disregarding timeouts). The full list of ontologies in K1 and K2 can be found in Table A.1 (appendix).

One interesting observation with respect to experiment run K1 is that there are cases where a modular technique succeeded in classifying the ontology within the given timeout, and the respective monolithic reasoner failed. 3.3% of the classification runs of modular techniques did not have a comparable run of a monolithic reasoner (same ontology, same delegate reasoner as monolithic reasoner). There are 12 such cases involving JFact, and 39 involving Pellet, scattered across 5 different ontologies (BIOMODELS, DRON, ICO, ONL-MSA, VSO), and no cases involving HermiT. Such cases exist involving all 6 modular reasoning techniques configurations (7 minus MON), the most successful ones being MOR with 16, MORA with 15 cases and MM with 6 cases.

$\mathcal{R}$	Tech	K2	K2 %	K1	K1 %
H	CC	105	100.00	42	87.50
H	CCO	105	100.00	42	87.50
H	CD	104	99.05	42	87.50
H	MM	99	94.29	33	68.75
H	MON	105	100.00	47	97.92
H	MOR	105	100.00	48	100.00
H	MORA	105	100.00	48	100.00
J	CC	105	100.00	30	62.50
J	CCO	105	100.00	30	62.50
J	CD	101	96.19	30	62.50
J	MM	102	97.14	27	56.25
J	MON	105	100.00	30	62.50
J	MOR	105	100.00	36	75.00
J	MORA	105	100.00	36	75.00
P	CC	102	97.14	17	35.42
P	CCO	102	97.14	16	33.33
P	CD	98	93.33	13	27.08
P	MM	102	97.14	15	31.25
P	MON	104	99.05	14	29.17
P	MOR	103	98.10	24	50.00
P	MORA	103	98.10	24	50.00

Table 7.3: Completion rates for all approaches. $\mathcal{R}$ is the reasoner (by first letter); Tech the technique or approach. See text for coding of techniques.

The variance of the OCT measurements was 5.61% and 3.64% on average, for K1 and K2 respectively, see Table 7.4. As can be seen by the (comparatively) low value for the third quartile (3rd Qu.), the majority of the variance comes from a few extreme outliers. In order for the analysis to not be too affected by these outliers, which may well be due to measurement error, we decided to *sample the median* (or closest to median) measurement and continue our analysis on those measurements only.

Cor	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
K1	0.11	0.87	1.94	5.61	4.26	137.80
K2	0.02	1.42	2.81	3.86	4.86	54.32

Table 7.4: Variance analysis of the Katana experiment. Metric: Overall classification time. Variance across multiple runs of the exact same experiment configuration, reported Coefficient of Variation (COV) in %. For example, the K2 OCT measurements varied on average by 3.64%.

Note that MORe does not always produce an $\mathcal{L}$-module, see Section 3.3.1. In our sample, this happens in 13 of the 35 ontologies: BCO, BDO, COGPO, DDI, EMO, FHHO, HEIO, HPIO, JERM, OBIWS, OMRSE, OPL, PCO. Similarly, the Atomic Decomposition sometimes consist of only a single connected component. This renders the CC and CCO strategies useless (and in some rare cases also the CD strategy). This happens in 23 out of the 35 ontologies: BCO, BDO, BHO, COGPO, DDI, DIKB, EMO, FHHO, GFO, GRO, HEIO, HPIO, HUPSON, JERM, KISAO, NGSONTO OBIWS, OMRSE, ONTOKBCF OPL, PCO, SPO, TMO. We therefore exclude from the following analysis all those cases for which the decomposition consisted of only a single chunk—the whole ontology—considering this a failure of the modular approach by default because it merely produces overhead without any possible gains (other than benevolent random effects). This "single-chunk problem" poses a serious limitation to the applicability of decomposition-based techniques as we investigate them in this chapter.

7.5.1 Overall Performance

Figures 7.2 and 7.4 show the final ranking of all techniques, broken down by reasoner and classification model (see Procedure 7.1 for details on the method). The ranking resulting from experiment run K2 in particular, Figure 7.4, shows that the monolithic approach outperforms the modular approaches in the majority of cases—regardless of the analytic model (blackbox, glassbox, etc.). This observation is strengthened by Figure 7.3 that shows how many times each technique took first, second and third place in the ranking (experiment run K1). Note that MOR, MORA, CC and CCO perform much worse in the ranking than one might expect in particular because of the large amount of cases where the the decomposition was of size one (and therefore excluded). Two modular approaches that do reasonably well in the ranking are MORA and MM. MORA, as a reminder, uses the MORe decomposition and classifies both modules using the primary OWL reasoner (for example HermiT). It is surprising that this approach does better than MOR, which uses ELK for the $\mathcal{L}$-module.[7]

A considerably different picture is revealed for the ranking by SST (sum of subsumption test times) rather than OCT, Figure 7.5 (K1) and Figure 7.6 (K2).

[7]We use version 0.4.1 of ELK, a release produced in August 2013. It is quite possible that this years release, 0.4.2, is considerably faster.

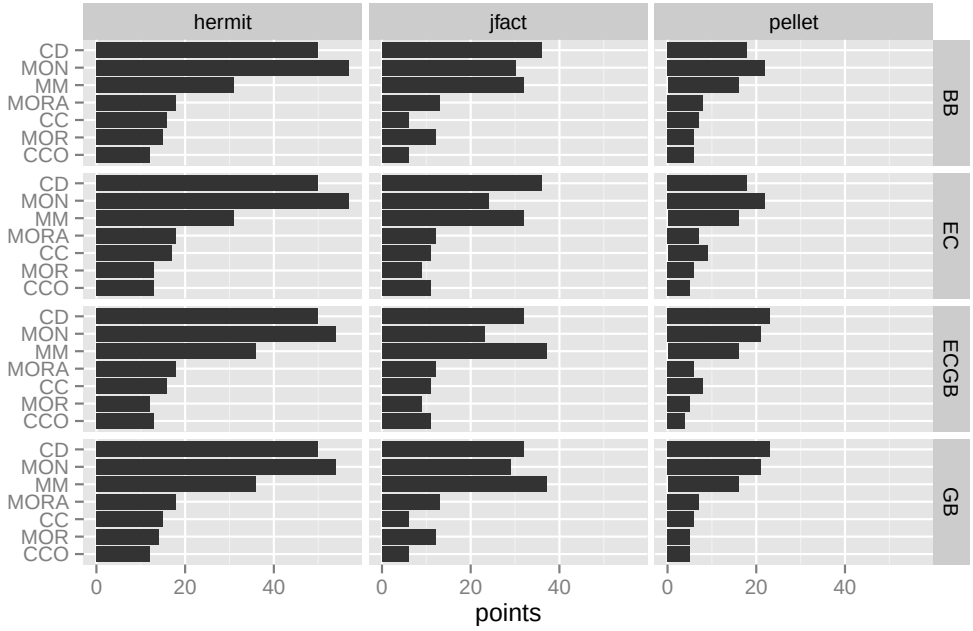


Figure 7.2: Ranking of approaches for experiment run K1, broken down by primary delegate reasoner and analytic model. The x-axis is the total number of points a technique scored based on our ranking method.

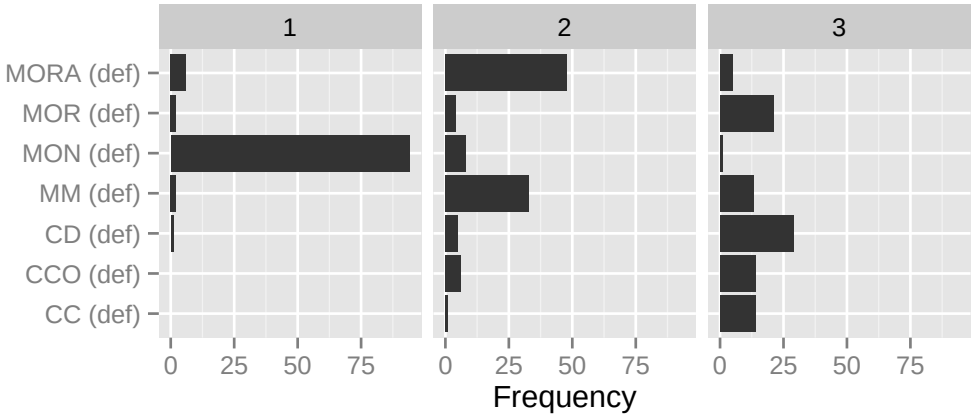


Figure 7.3: Number of times an approach has taking first, second or third position in the ranking (blackbox model, experiment run K1).

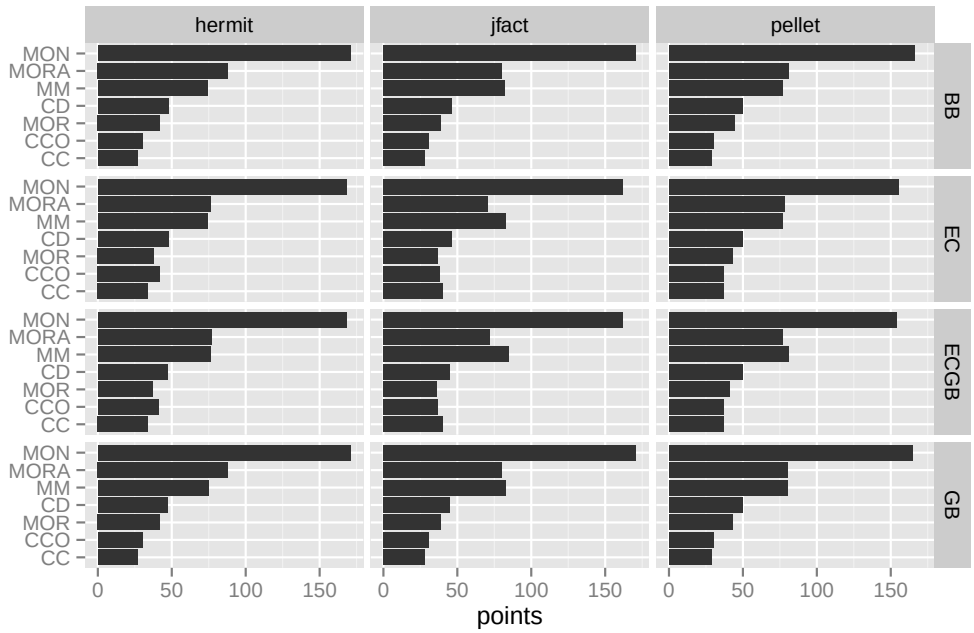


Figure 7.4: Ranking of approaches for experiment run K2, broken down by primary delegate reasoner and analytic model. The x-axis is the total number of points a technique scored based on our ranking method.

While, considering the blackbox model, monolithic reasoning is faster than modular techniques for HermiT and Pellet, the differences are considerably smaller, and, for JFact, both CD and MM do better in terms of SST than the respective monolithic strategy. Considering the glassbox model, the picture looks even better for modular techniques: the sum of subsumption test times for MM and ST is often less than the comparable monolithic reasoner, at least in the cases of Pellet and JFact. Again, the connected component based decompositions and the MORe decomposition-based approaches suffer from the many exclusions due to single-module decompositions. We will see later that in terms of subsumption test hardness reductions, modular approaches nearly always outperform the monolithic ones. Note that when using HermiT (Novel Approach, hyper-tableau based), modular techniques do not seem as effective as when using Pellet and JFact, the two Enhanced Traversal, tableau-based OWL reasoners. This can have a number of explanations. For example, Hermit's traversal might be more efficient, which reduces the effect of modular techniques on traversal space pruning; or HermiT might simply be, as we have concluded from Chapter 6, less sensitive to modularly irrelevant axioms, therefore reducing the benevolent effect

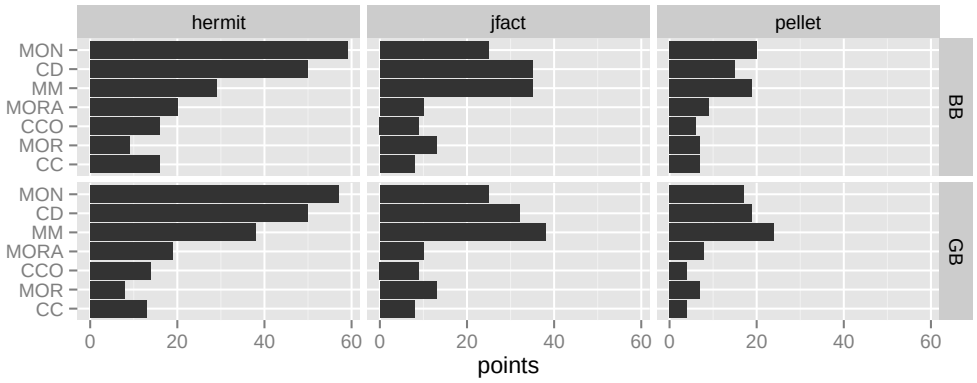


Figure 7.5: Ranking of approaches based on SST, for experiment run K1, broken down by primary delegate reasoner and analytic model. The x-axis is the total number of points a technique scored based on our ranking method.

of modularity on subsumption test hardness.

It is also important to remember at this point that results for Pellet with respect to subsumption test duration and counts are influenced by the fact that it can potentially fall back on its internal $\mathcal{EL}$-reasoner. While this does not happen for the whole ontology, it is quite possible that this happens for individual chunks in the decomposition. That means that a particular subsumption might be derived by the internal reasoner, but show up in this analysis as *avoided*. In these cases, the effect of modularity would not be to prune the search space, but to replace potentially hard SAT tests by a cheaper alternative (in this case a sequence of derivation rules). It remains to be a part of future work to include the measurements of derivation rules triggered by the internal $\mathcal{EL}$-reasoner.

7.5.2 What Stages in the Reasoning Process Contribute to OCT?

Table 7.5 breaks down the contribution of sub-processes to the overall classification time (OCT). The first thing to note, and also the reason why a breakdown by delegate reasoner was necessary, is that the relative impact varies significantly across primary delegate reasoners. For example, the sum of pre-reasoning processing times of all delegates is generally higher for HermiT than for the other two reasoners, because this stage is more dominant in HermiT per se: during PRP, HermiT performs satisfiability tests of the leaf-nodes in the class graph, work that the other reasoners have to do during the ST stage. This can be observed

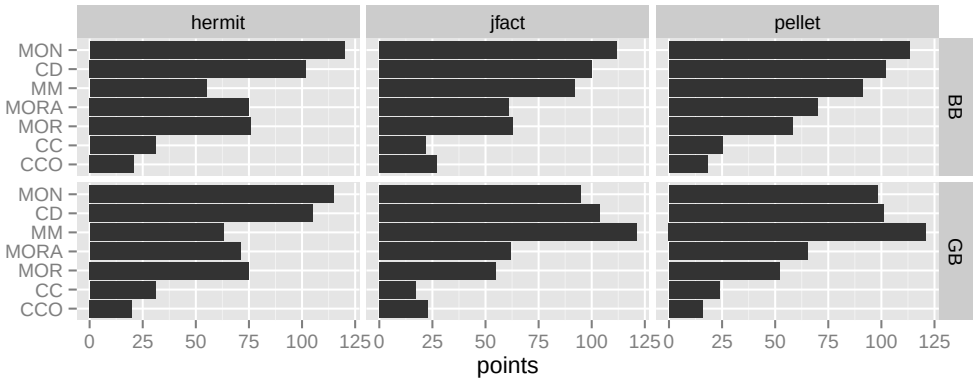


Figure 7.6: Ranking of approaches based on SST, for experiment run K2, broken down by primary delegate reasoner and analytic model. The x-axis is the total number of points a technique scored based on our ranking method.

in Table 7.5: for example, monolithic (MON) HermiT (H) spends 46.9% of the classification time during pre-reasoning processing (PR in table), and only 20.0% in the traversal stage (ST), while Pellet (P) spends 0.2% only in PR and 42.3% in ST.

The impact of decomposition time is, unsurpringly, generally higher for approaches based on the Atomic Decomposition (CC and CCO). In the case of CD, that impact is shifted to another part of the pre-processing phase (M1), which is dominated by the extremely inefficient implementation of the community detection algorithm. In the case of MM, the relative impact of the decomposition is, most likely, diminished by the overhead induced by the overlap of maximal modules and a potentially large amount of delegates (see Section 7.2.1 for more about these threats). The contribution of the decomposition time to the overall classification time is probably most informative for the CCO strategy, as this strategy typically involves little redundancy and little overhead because of delegate reasoner creation (compare also Table 7.7). For our K2 corpus, the decomposition (for CCO) takes on average between 19% and 37% of OCT (depending on the primary delegate reasoner); for the K1 corpus between 16% and 28%.

The high values of the second *Res* column (second to last in Table 7.5) for approaches employing ELK for EL modules reflect the time spend by ELK doing classification (in our case only MOR). This shows that ELK does in fact most of the work during the MOR classification.

$\mathcal{T}$	$\mathcal{R}$	M1	DC	Res	M2	PP	CC	PR	ST	PO	Res	M3
CC	H	42.2	36.6	5.6	57.6	11.6	0.3	20.9	17.6	7.0	0.3	0.2
CC	J	31.8	27.7	4.1	68.1	28.7	0.1	0.0	33.2	5.8	0.3	0.1
CC	P	21.1	18.7	2.4	78.8	38.4	1.0	0.3	35.1	3.6	0.4	0.1
CCO	H	42.7	36.2	6.5	57.1	9.7	0.1	21.6	18.5	7.3	0.1	0.2
CCO	J	32.3	27.5	4.8	67.6	28.8	0.1	0.0	32.9	5.7	0.0	0.1
CCO	P	22.1	19.0	3.0	77.8	37.1	1.0	0.2	36.0	3.4	0.0	0.1
CD	H	91.4	4.5	86.9	8.6	0.8	0.1	2.9	4.3	0.5	0.0	0.0
CD	J	87.1	3.7	83.4	12.9	3.4	0.2	0.0	8.9	0.4	0.0	0.0
CD	P	82.6	4.3	78.3	17.4	4.6	1.9	0.0	9.3	1.6	0.0	0.0
MM	H	13.7	9.7	3.9	86.3	25.7	4.0	21.0	24.9	9.8	0.8	0.0
MM	J	13.7	10.0	3.7	86.2	41.2	1.6	0.1	33.3	7.9	2.1	0.0
MM	P	9.0	6.6	2.5	90.9	32.7	7.0	0.2	38.3	10.9	1.8	0.0
MON	H	4.7	0.2	4.5	95.0	15.4	3.1	46.9	20.0	9.5	0.0	0.3
MON	J	2.5	0.1	2.4	97.3	41.1	1.6	0.0	48.9	5.8	0.0	0.2
MON	P	1.0	0.0	0.9	98.9	45.4	4.5	0.2	42.3	6.4	0.0	0.1
MOR	H	6.2	5.8	0.4	93.7	3.2	0.0	12.5	8.3	2.0	67.6	0.1
MOR	J	5.7	5.3	0.4	94.2	12.5	0.2	0.0	16.4	1.7	63.4	0.1
MOR	P	3.9	3.6	0.3	96.0	23.0	0.6	0.1	26.7	1.4	44.2	0.1
MORA	H	19.8	18.2	1.6	79.9	13.8	0.4	24.2	30.1	11.3	0.1	0.3
MORA	J	13.9	12.8	1.1	85.9	37.2	0.3	0.0	41.9	6.6	0.0	0.2
MORA	P	7.1	6.6	0.5	92.8	45.3	1.1	0.2	41.4	4.6	0.0	0.1

Table 7.5: Impact of sub-processes in % of overall classification time (OCT), experiment run K2. The columns in gray represent the three main modular classification stages: M1 pre-processing, M2 modular classification and M3 post-processing. DC is the decomposition time, columns PP, CC, PR, ST and PO stand for the sum of the contribution of the delegate reasoners: PP is the sum of pre-processing stages of all delegates (in % of OCT), CC the sum of all consistency checks, PR the pre-reasoning processing, ST the time spend traversing the class graph and PO the sum of all post-processing time measurements. The two columns labelled *Res.* represent the remaining time not accounted for by the preceding columns with respect to M1 and M2 respectively.

Figures 7.7 (K1) and 7.8 (K2) show the performance profiles of three example ontologies, i.e. the break down of sub-processes contributing to the overall classification time. The full set of all ontologies can be found in the appendix (Section A.4). The x-axis represents total classification time; the actual values are not of interest (only the ratio between the techniques), and the axis is re-scaled for every sub-plot. The BT ontology in Figure 7.7 is a quite unusual case. As can bee seen, OCT is heavily dominated by subsumption test time (in the case of HermiT, including pre-traversal, as described before). Apart from MM, there are virtually no differences between the different approaches. MM is beneficial using JFact, and detrimental using HermiT. BT is an example where decomposition time has no impact on OCT. A decomposition with small chunks (redundancy

mitigated) might have a significant effect on OCT by pruning the search space and making tests easier.

Another interesting example is NPO (same figure as before). NPO is a very clear example of where modular approaches fail due to the strong impact of the decomposition time (DEC), which all by itself takes much longer (HermiT and Pellet) or almost as long (JFact) as it took the monolithic approach to classify the ontology entirely. The MORe approaches are not as detrimental due to the faster decomposition, but are generally doing worse than the monolithic approach in all cases. STATO[8] shows a different picture depending on the primary delegate: using HermiT, the monolithic approach does considerably better than the modular one, using JFact, the picture is inverted: MM (both EL and default variant) does almost three times better than the monolithic variant (otherwise exhibiting a similar profile in terms of relative impact of sub-stages).

Figure 7.8 shows three examples from K2. Without going much into depth, we want to point out that many of the ontologies in K2 have a relatively low impact of SST. In Table 7.6 we can see a breakdown of the contribution of SST to OCT (mean), by technique and primary delegate reasoner. Overall, the impact of subsumption testing in K1 contributes to OCT rather moderately: 15%. CAO is a notable exception: here, subsumption testing is dominant, regardless of the modular approach. Like STATO, it is known to cause problems to reasoners [LMPS15]. Modular approaches using HermiT do a bit better on CAO than monolithic HermiT, and similar to BT (Figure 7.7), the overhead of computing the decomposition does not play any role. This suggests that benefits from modularity are related to the impact of subsumption test time. K1 provides some further evidence to this conjecture: subsumption testing accounts on average for 47% of overall classification time and modular approaches did considerably better in experiment run K1 than in experiment run K2.

The last factor of interest with respect to contributions to overall classification time is the contribution of the hardest delegation. Figure 7.9 shows the distribution of relative impact of the hardest chunk broken down by modular technique. In general, the stronger the possibility for a decomposition to contain a large chunk (that needs to be processed by a full fledged OWL reasoner), the more the central tendency of the distribution shifts towards the right. For example, chunks in the MM-decomposition are generally quite small; therefore, the vast majority

[8]STATO is known to cause problems for reasoners [LMPS15].

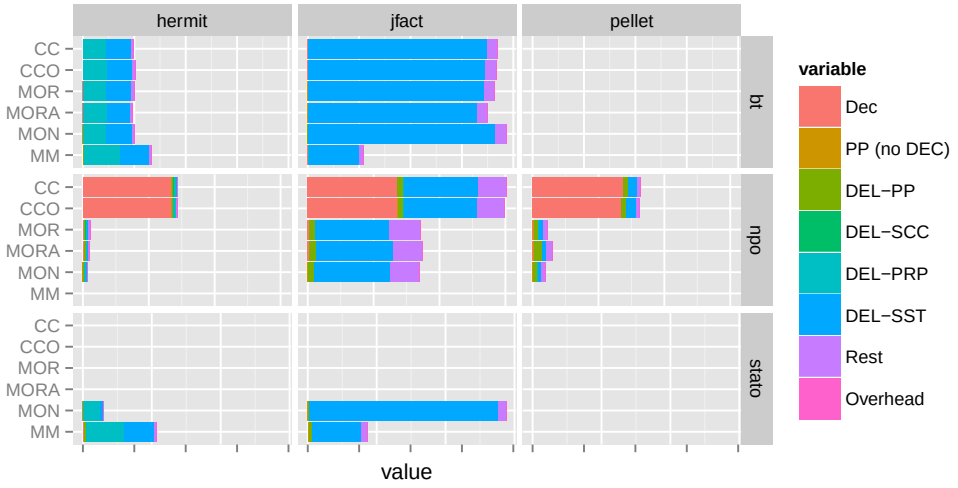


Figure 7.7: Performance profiles of three example ontologies from experiment run K1. x-axis is time spent, the full extent of the bar represents the OCT, ratios between times being preserved across ontologies. As we are only interested in comparing ratios, we omitted the axis labels. Dec is the decomposition time, PP (no DEC) is the remaining time needed for preprocessing (assigning ontologies to delegate reasoners, determining order), DEL-PP is the total time spent by delegate reasoners doing pre-processing, DEL-SCC is the total time spend consistency checking, DEL-PRP pre-reasoning processing and DEL-SST subsumption testing. Overhead is the time spend recording subsumption tests and other meta-data during classification. Note that we have excluded the CD strategy on all plots because it was dominated by PP (no DEC) and rendered the other techniques unreadable.

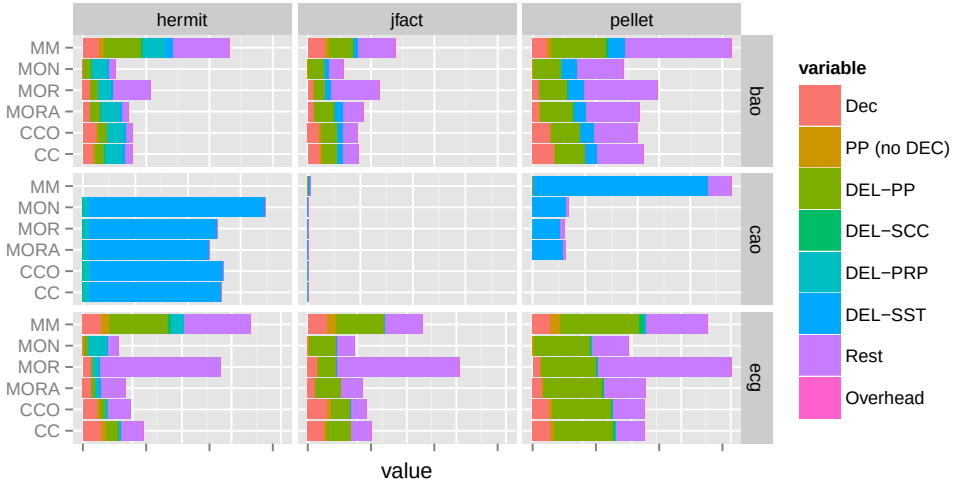


Figure 7.8: Performance profiles of three example ontologies from experiment run K2. See caption of Figure 7.7

$\mathcal{R}$	Tech	K1	K2
HermiT	CC	25.13	12.56
HermiT	CCO	23.42	12.69
HermiT	CD	26.29	13.59
HermiT	MM	39.04	12.71
HermiT	MON	21.59	12.69
HermiT	MOR	20.93	5.52
HermiT	MORA	20.93	11.59
JFact	CC	76.27	26.68
JFact	CCO	76.08	26.33
JFact	CD	72.81	26.21
JFact	MM	66.23	19.60
JFact	MON	75.19	26.76
JFact	MOR	50.08	10.15
JFact	MORA	52.22	18.40
Pellet	CC	50.92	11.51
Pellet	CCO	51.37	12.13
Pellet	CD	41.85	13.12
Pellet	MM	29.15	9.61
Pellet	MON	51.20	14.04
Pellet	MOR	16.26	6.62
Pellet	MORA	19.93	8.13

Table 7.6: Contribution of subsumption testing (SST) to OCT (blackbox model) in %. For example, for the CC approach, using HermiT as primary delegate, subsumption testing took up, on average, 12.56% in K2.

of hardest delegations contribute less than 25% to the overall classification time (note the log scale). In the cases of MOR and MORA, we can also observe a hardness shift: because MOR will process the large $\mathcal{L}$-module with ELK (more efficient), it will be less dominant on OCT. Note that it is quite possible that the hardest delegation in the MOR case is, occasionally (remember the mean relative impact, Table 7.5), the classification of the generally smaller remainder module. In the MORA case however, the $\mathcal{L}$-module will be processed by the full fledged OWL reasoner (JFact, Pellet or HermiT). As the $\mathcal{L}$-module is typically a quite large fraction of the ontology, classifying it with the more inefficient reasoner will make the $\mathcal{L}$-module classification be a much larger proportion of the OCT. In Figure 7.9, this effect can be observed by the distribution shifting considerably to the right from MOR to MORA.

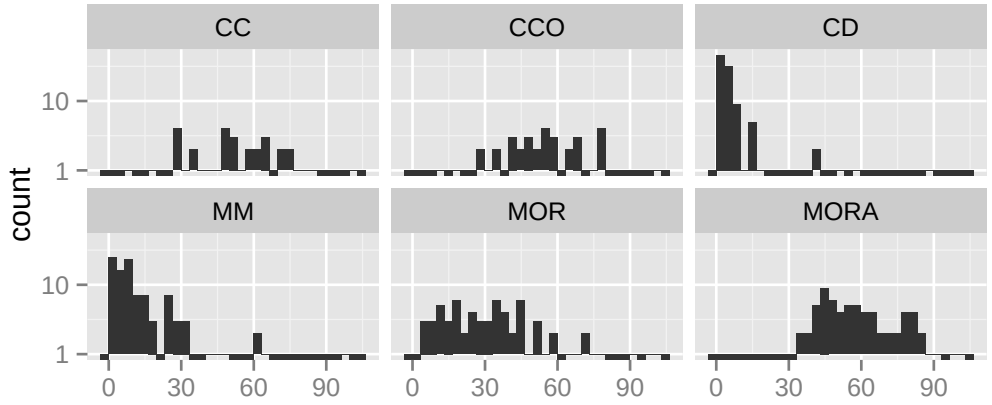


Figure 7.9: Histogram of the contribution of the hardest chunk in the classification OCT in %.

7.5.3 Test Avoidance and Redundancy

In the following, we will analyse the effect of modularity on test avoidance and redundancy. We consider 402 comparisons: the number of modular approach/delegate/ontology combinations that (1) successfully classified and (2) have a decomposition containing at least 2 chunks. As a reminder: avoided tests are those triggered by the monolithic reasoner that are not triggered by the modular technique and extra tests are those triggered by the modular approach not triggered by the monolithic approach (Definition 7.2); common tests are triggered by both the monolithic and the comparable modular approach. Redundant tests are those triggered by the modular approach more than once, and unique tests are the hardest instances of single test triggered during a classification run (Definition 7.1); Figure 7.10 breaks down the approaches by whether they are generally avoiding tests, generally producing extra tests, or whether extra tests and avoided tests are generally cancelling each other out. The coding was explained in depth in Section 7.3.2.

Considering the blackbox analytic model, we see that CD and MM have the potential to prune the traversal space the most—but also have the most detrimental cases of any technique by far. Overall, there are 12 cases where modular approaches significantly prune the search space (high magnitude), 1 case of CC, 4 cases of MM, 4 cases of MOR and 3 cases of MORA. 4 cases related to the RXNO ontology, 2 to the ECG, WB-BT, NGSONTO ontologies, and 1 to ECO and FHHO. The extreme cases of extra tests were all caused by the MM strategy

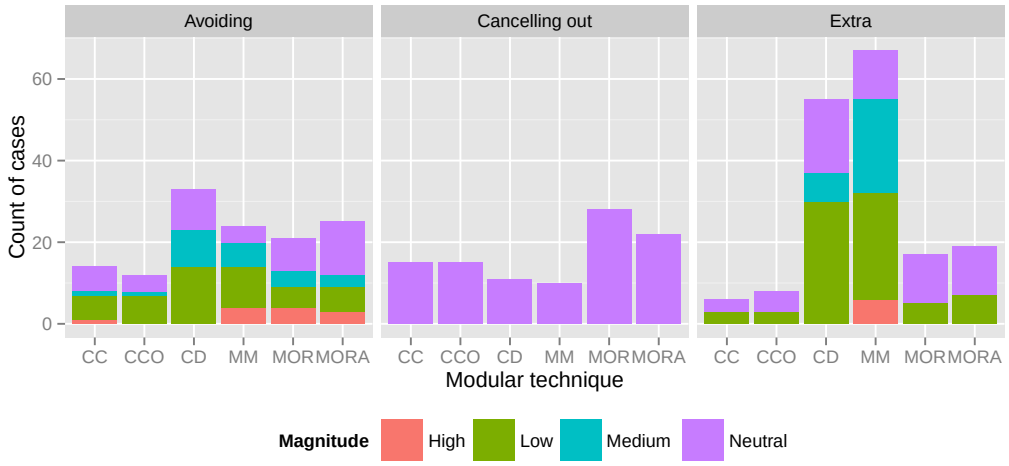


Figure 7.10: Break down of cases by whether they are predominantly avoiding tests, producing extra tests, or both extra and avoided tests cancelling each other out, further broken down by the magnitude of the effect (blackbox model). For example, there are 33 cases (classification runs) for which CD has helped avoiding subsumption tests. Out of these, 9 have a medium magnitude of avoidance, 14 have a low magnitude, and the remaining 10 are neutral.

on the following ontologies: BHO, EMO, GFO, HUPSON, OPE and SAO. It is very possible that the average size of the chunks in the decomposition have something to do with the potential to prune: For MM, the average size of the chunk is only about 100 axioms, for CC 379 (note the large standard deviation), for CD 2,120, for CCO 2,237, for MOR and MORA 2295 and for MON (the monolithic approach) 4,771.

For the cancelling out cases, the vast majority (83%) have neither extra nor avoided tests, and therefore an avoidance score of 0. There are 12 cases with one extra, one avoided test, and 5 cases with equal numbers of tests in both categories higher than 1.

Figure 7.11 shows the same analysis, while ignoring redundant tests (glassbox model). While the cases where avoided and extra tests are cancelling each other out remain mostly unchanged (all except for a single one), 21 cases (12.2%) of all the 172 *extra* cases under the blackbox model now move to the *avoiding* group. 12.4% out of the 129 avoiding cases under the blackbox model changed to a higher category in terms of magnitude. All 5 newcomers to the avoiding category with high magnitude were cases of MM. Altogether, the number of cases in which there are potential gains in terms of traversal space pruning only barely outweigh the

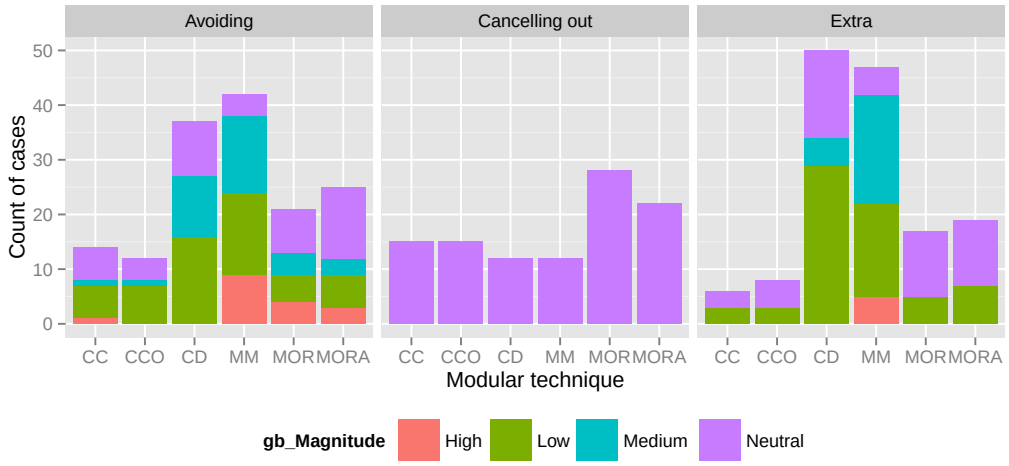


Figure 7.11: Break down of cases by whether they are predominantly avoiding tests, producing extra tests, or both extra and avoided tests cancelling each other out, further broken down by the magnitude of the effect (glassbox model).

number of cases that are detrimental (by 151 to 147). This suggests that at least in the case of decomposition-based classification, it is quite likely that heuristics are needed to determine the applicability of a modular approach upfront (similar to the mechanism that decides whether the internal $\mathcal{EL}$ classifier of a reasoner can be used).

The effect on the traversal space does not necessarily have any great consequences on the overall classification time. Figure 7.12 directly corresponds to Figure 7.10. Instead of numbers of tests however, we visualise the effect in terms of proportion of OCT. The metric is computed by the following formula: $\frac{ATT-ETT}{OCT}$. The vast majority of cases only have low to no (neutral) impact. Beneficial (avoiding) cases of medium and high impact constitute only about 4.7% of the cases overall. In contrast to that, 14.7% of the high and medium impact cases are beneficial in experiment run K1 (10.7% being of high magnitude). We conclude from these observations two things: (1) avoided or extra tests do not necessarily make a difference to overall classification time and (2) for some ontologies using the decomposition is beneficial, for some other ontologies detrimental—modular techniques do *not* generally prune the search space.

Table 7.7 shows the mean values of some of the key metrics in this section. BBA and GBA are the mean values of our avoidance score, for the blackbox (BB) and glassbox (GB) models respectively. Negative values indicate extra tests, and

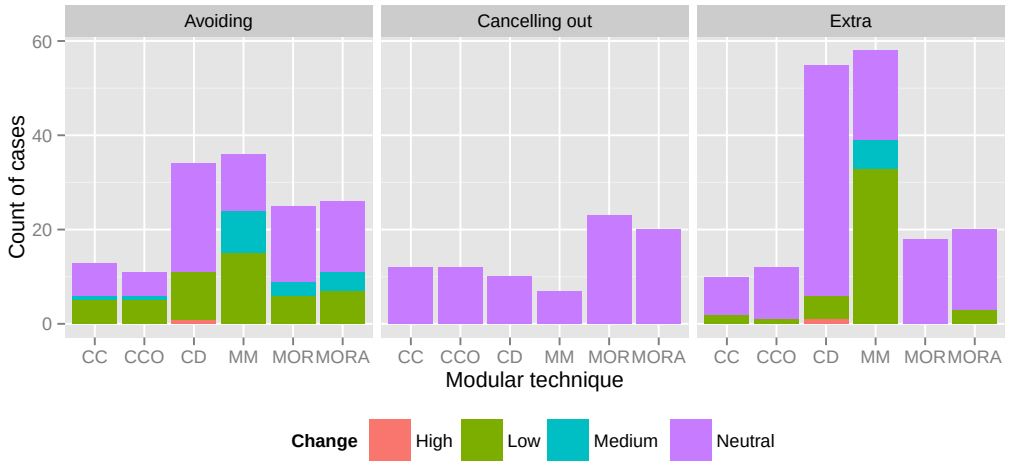


Figure 7.12: Impact of avoidance on OCT (K2), quantified as the total time of avoided tests (ATT), minus the total time of extra tests (ETT), divided by overall classification time (OCT). The magnitude of the effect is high, if it higher than 50% of the overall classification time, medium if it is between 10% and 50%, low if it is between 1% and 10% and neutral if it is less than 1%. For example, there is one case of CD for which the difference of avoided to extra tests accounted for more than 50% of the OCT (high magnitude).

positive values subsumption test avoidance. As we have pointed out before, the differences across ontologies are great; mean values should be read with caution. The AV, COM and EX should give a sense of the average proportions of avoided/common/extra tests. All three metrics are represented as ratios of STC-MOD (total subsumption test count of modular reasoner). As can be seen, common tests dominate in most cases. In the cases where they do not, extra tests are typically inflated because of redundancy: A subsumption test triggered by the modular reasoner multiple times, which has not been triggered by the monolithic reasoner. In terms of redundancy (RED), the CD (around 24%) and MM (around 77%) approaches are dominating by far. It is obvious that despite having typically much smaller modules in MM than in CD the redundancy is significantly higher. This suggests considerable overlap between some of the chunks in the decomposition. It is interesting that the CC and CCO strategies do not generate any redundancy at all. This is probably due to the fact that both decompositions are partitions, i.e. all chunks in the decomposition are mutually non-intersecting. The very low redundancy of the MORA approach is most likely due to the fact

that the remainder module is typically small. The fact that Pellet has a predominantly large proportion of common tests suggests that it typically did *not* fall back into the internal $\mathcal{EL}$-reasoner for dealing with the $\mathcal{L}$-module.

$\mathcal{R}$	MT	BBA	GBA	AV	COM	EX	RED	UN
HermiT	CC	-3.56	-3.56	9.49	86.60	13.40	0.00	100.00
JFact	CC	-0.01	-0.01	6.28	93.72	6.28	0.00	100.00
Pellet	CC	35.74	35.74	38.98	89.00	11.00	0.00	100.00
HermiT	CCO	18.24	18.24	29.41	86.81	13.19	0.00	100.00
JFact	CCO	-4.17	-4.17	5.58	91.63	8.37	0.00	100.00
Pellet	CCO	26.55	26.55	33.15	87.38	12.62	0.00	100.00
HermiT	CD	-67.52	-40.66	19.64	60.77	39.23	22.61	77.39
JFact	CD	25.51	31.59	33.67	91.91	8.09	24.89	75.11
Pellet	CD	69.42	75.63	67.90	76.27	23.73	24.85	75.15
HermiT	MM	-951.39	-517.08	3.22	33.22	66.78	74.82	25.18
JFact	MM	30.03	513.40	53.61	80.60	19.40	78.06	21.94
Pellet	MM	49.76	2937.63	165.51	76.47	23.53	77.60	22.40
HermiT	MOR	113.88	113.88	76.99	75.82	24.18	0.00	100.00
JFact	MOR	97.89	97.89	100.32	96.44	3.56	0.00	100.00
Pellet	MOR	366.92	366.92	186.62	93.67	6.33	0.00	100.00
HermiT	MORA	113.09	113.05	73.10	74.56	25.44	0.31	99.69
JFact	MORA	45.18	45.18	52.41	96.53	3.47	3.89	96.11
Pellet	MORA	367.31	367.36	192.94	93.15	6.85	0.04	99.96

Table 7.7: Avoidance and redundancy analysis of K2. BBA is the mean avoidance score using the blackbox model. A positive number indicates that avoided tests outweigh extra tests. The closer the value to 0, the less magnitude that effect has. GBA is the same, recalculated under the glassbox model. AV, COM and EX are the proportion of avoided, common and extra tests to the total number of tests (in %) as measured by the modular reasoner (blackbox model). RED and UN are the respective proportions of redundant and unique tests.

As a side observation, there are 6 cases in which the monolithic reasoner triggered a test, and the comparable modular technique did not trigger any tests at all (4 MM, 1 CC, 1 CD). This number is not significant (1.5% of the cases), but it is interesting to know that this effect can occur at all. There were 4 cases for HermiT and 2 cases for Pellet. It is possible that both were able to use their internal deterministic engines, without the need to trigger any tests.

7.5.4 Test Hardness

Table 7.8 shows the effect of the decomposition on the hardness of subsumption tests. It is striking that in nearly all cases, the decomposition was beneficial to the hardness of the tests triggered by both the monolithic and the modular reasoner (which constitute the majority, see Table 7.7). Detrimental effects can be found only for the MM strategy (most likely due to duplicate tests), and only under the blackbox model. Among the cases with changes in hardness of common tests of more than a 10-fold we can find all three delegates, all using the MM strategy, and scattered across 8 different ontologies. The magnitude varies a lot across primary delegate and modular approach. In general, these results confirm the observation in Chapter 6 that subsumption test hardness is, on average, reduced under modularity. The occasionally much higher effect witnessed in this chapter might be due to the fact the modules in the decompositions we have investigated are typically smaller than the ones sampled by our subset-signature approach (Section 6.3.1).

Modular techniques have a, occasionally great, potential to reduce the average hardness of subsumption tests. In only 4 out of 18 configurations (delegate reasoner - modular approach pair) did the decomposition have a detrimental effect on mean test hardness, all of which involve modular techniques with potentially large chunks (CC ,CCO, MOR). 3 of the configurations involve Pellet and one HermiT. Again, we have that, the smaller the chunks (MM,CD), the more potential for hardness reduction. The MM strategy is able to, on average, reduce the mean hardness of tests by more than 200%. Note that it is possible that the magnitude of this effect was caused by replacing hard subsumption tests with (possibly harder) alternative tests conducted by the internal non-deterministic reasoners, which our framework ignores; however, given that JFact exhibits a similar profile without having an internal consequence based reasoner, we do not expect this to happen often.

$\mathcal{R}$	Tech	COM	STM	COMU
HermiT	CC	-15.15	-14.01	-15.15
HermiT	CCO	-9.32	7.11	-9.32
HermiT	CD	-21.07	-19.55	-37.00
HermiT	MM	89.00	-166.55	-73.06
HermiT	MOR	-22.12	-1.21	-22.12
HermiT	MORA	-40.75	-13.30	-42.57
JFact	CC	-1.06	-1.09	-1.06
JFact	CCO	-6.60	-6.42	-6.60
JFact	CD	-5.63	-22.36	-23.06
JFact	MM	22.18	-204.54	-36.04
JFact	MOR	-5.25	-3.81	-5.25
JFact	MORA	-1.44	-4.66	-3.54
Pellet	CC	-39.50	15.42	-39.50
Pellet	CCO	-25.48	14.14	-25.48
Pellet	CD	-44.55	-22.24	-57.00
Pellet	MM	56.41	-238.04	-137.81
Pellet	MOR	-2.86	5.97	-2.86
Pellet	MORA	-13.63	-0.32	-13.63

Table 7.8: Effect of decomposition on subsumption test hardness (K2). COM is the fold change of the total time of tests triggered both by the monolithic and the modular approach. STM is the (respective) fold change of mean test hardness (taking into account all tests). COMU is the fold change of common test under the glassbox model (redundant tests ignored). A positive value (in all cases) indicates a detrimental effect of the modular technique on the measure. The fold change is, due to some extreme outliers, aggregated as median for all three measures and here reported in %.

7.6 Discussion

Blackbox decomposition based approaches rarely outperform their monolithic counterparts. However, the applicability of blackbox decomposition-based reasoning based on syntactic locality-based modules differs extremely across ontologies. Some ontologies, typically involving a lot of subsumption testing, can benefit from the decomposition. There are even a few rare case where a modular approach was able to classify, while the comparable monolithic strategy was not.

One of the strongest arguments against employing the CC, CCO and MORe-based strategies is their limited applicability. The *single chunk problem* renders any efforts in extracting modules totally useless; we therefore judge them as failures by default. CC and CCO were only applicable to less than 35% of the

ontologies in K2, and the MOR and MORA approaches failed by default in more than 37% of the cases. Note that the single chunk caveat only applies to the specific approaches we have investigated as part of this chapter.

What are the main reasons for modular approaches failing to be beneficial? Initially, we expected the decomposition time to contribute considerably more to the overall classification time. This was the motivation for the encoded chunking (EC) analytic model. From Figures 7.2 and 7.4 we could see that presuming a somehow encoded decomposition did not make a big difference at all. We also know from email communication[9] that the optimised Atomic Decomposition as implemented in the latest releases of FaCT++ can be up to 10 times faster than the (experimental) implementation as part of the OWL API tools. At least for K2, this would mean that decomposition time could be reduced to less than 3-5% of OCT–better than anticipated. However, from experience we know that it can take many hours, even days, to compute the Atomic Decomposition on some of the larger (and often harder) ontologies. More experiments need to be carried out to cover ontologies of all sizes and difficulties.

The second main threat to the application of decomposition-based classification approaches is the redundancy induced by the overlap between chunks. As part of this chapter, we have analysed the redundancy of subsumption testing. Our main conclusions are that there are cases (a modular approach using a particular primary delegate on some ontology) for which the modular technique proved highly beneficial with respect to test avoidance, and some others where it proved highly detrimental. We will, as part of future work, improve our characterisation of beneficial and detrimental cases and try to develop heuristics that can determine the applicability of a modular decomposition to improve reasoning time upfront. Most likely, these heuristics will be based on the logical structure of the ontology, its size, potential axiom interactions and its expressivity. Note however, that subsumption testing is by no means the only source of redundancy. In fact, as can be seen in Section A.4 (appendix), many classifications have a considerable impact of delegate pre-processing time. It is quite likely that a lot of what is done during pre-processing, depending on the reasoner (for example normalisation, told subsumers or absorption) is repeated across multiple chunks. Some cases are even effected by a significant contribution of the (totally redundant) consistency check that monolithic reasoner conducts before classification. It

[9] Dmitry Tsarkov and Ignazio Palmisano.

remains to be seen whether it is worth adjusting the implementations of delegate reasoners in such a way that these redundancies can be avoided (shared pre-processing, shared partial classifications, shared knowledge of consistency, etc.) or whether it will be necessary to implement a fully modularity-sensitive OWL reasoner, see Section 3.1.1, instead.

7.6.1 Methodological Reflections

The choice of the K2 corpus as the major point of analysis was a difficult, and questionable one. Biasing the sample to those ontologies that all reasoners have successfully dealt with (within a timeout of 1 hour) will necessarily create a bias towards easier ontologies—for harder ontologies it is more than likely that at least one reasoner is not able to deal with them in time. But, as we saw from our comparisons with the K1 corpus, modular techniques seem to be more effective in the context of harder ontologies. Therefore, the K2 analysis might emphasise the detrimental effects of modular techniques. The next step in the course of our investigation will include an in-depth analysis of the nature of ontologies that are more agreeable to decomposition based reasoning.

One interesting thing to note is that the variance between our Amazon EC2 run K2 and our own Mac Mini cluster run K1 appears mostly similar. We expected the variance on the Amazon cloud to be much worse. While the median variance is slightly higher for the Amazon run K2, the mean is slightly lower. At a closer look, two of the top four classifications with the worst variation in K1 relate to a single ontology, PORO (and both HermiT and JFact), which suggests that the source of the variance could also be the ontology itself.[10] In K2, the top three cases of variance relate to the same ontology (OPL), with different delegates (Pellet an JFact) involved, and all different modular techniques. Based on these observations, we make the preliminary suggestion that running reasoning time experiments on the cloud is safe (from the variance point-of-view). However, high variance can have consequences on the conclusions to be drawn. In this thesis, we only ever checked the variance to convince ourselves that it is acceptable overall. In the future, we want to take into account the variance more systematically by creating a variance score per ontology that can be used across experiments.

[10]PORO is known to cause problems for reasoners from previous experience.

7.7 Summary of Key Observations

In the following, we will summarize the key observations made as part of this chapter.

- Decomposition-based approaches are not generally suitable to improve classification performance.
- In some cases, modular approaches outperform their monolithic counterparts.
- Suitability to improve performance depends largely on the ontology. A preliminary indicator for suitability is the impact of subsumption test hardness to overall classification time.
- Reasoning with decompositions rarely has a detrimental and frequently a very beneficial effect on subsumption test hardness. We confirmed our observation in Chapter 6 that, on average, test hardness is reduced under modularity. Our results suggest that the smaller the module, the stronger the effect of hardness reduction.
- Whether reasoning with decompositions has a detrimental or beneficial effect on subsumption test avoidance depends heavily on the particular ontology at hand. The techniques with the largest potential for avoidance for some ontologies (CD, MM) also have the largest potential for extra tests on others. A lot, but not all, of the detrimental cases are caused by the occasionally great amount of redundancy.
- JFact is considerably more amenable to decomposition-based reasoning than HermiT and Pellet.

Chapter 8

Conclusions

In this thesis, we have investigated the use of modules and module-based decompositions to optimise OWL classification. In particular, we have shown that:

- Syntactic locality-based $\bot$-modules are occasionally harder than their parent ontology. Such pathological modules are typically, but not always, large fractions of the parent ontology.
- The impact of subsumption testing is significant only for a small proportion of BioPortal, which threatens the applicability of modular approaches to optimise classification by reducing test hardness.
- Using modules and module-based decompositions reduces the average hardness of subsumptions tests. The magnitude of the effect differs significantly by reasoner and modular technique employed. However, as subsumption testing is only occasionally a major contributor to overall classification time, the overall savings due to reduced test times are not always significant.
- The effect of modular decompositions on search space pruning depends heavily on the ontology. Modular techniques have the potential to prune the traversal space significantly, but can also induce significant amounts of extra tests. Further efforts are needed to characterise the ontologies which are likely to benefit—and which are likely not to.
- Contrary to our expectations, decomposition time does not generally constitute the majority of the overhead of decomposition-based classification approaches. Redundancy is often a more dominant problem.

Furthermore, we have:

- Designed a framework (Katana) for benchmarking decomposition-based classification approaches.
- Designed a benchmark and analysis pipeline that allows a more fine grained view of traversal-based classification approaches (OWL Reasoner Stage Benchmark).

- Designed a number of novel methods to measure the effects of modularity on OWL classification.
- Created, published and characterised a large dataset with fine grained timings of classification time, including sub-processes and individual subsumption test times, as a basis for furthering the understanding of modular and monolithic classification.

8.1 Summary of Contributions

In this thesis we set out to investigate the potential of using locality-based $\bot$-modules to make classification easier, specifically by reducing the hardness of subsumption tests or avoiding them altogether. We have not conclusively answered the question of whether modules are generally beneficial. We rooted our benign module conjecture (no module is ever harder than its parent ontology) in the fact that any module contains all the reasons for a subsumption to hold between classes in its signature. In practice, this conjecture turned out to be false, casting some initial doubt on the usefulness of modularity for classification. However, as such pathological cases occurred only rarely, and we were able to observe the protective effect of modularity in general—random subsets that are harder to classify than the ontology became easier than the ontology when "modularised"—we continued our investigation to determine the effect of modularity on subsumption test hardness as triggered during classification. We showed that only a minority of the ontologies in BioPortal are actually amenable to subsumption test hardness reduction: most ontologies in the corpus are fairly inexpressive and did not require reasoners to trigger subsumption tests at all. We continued to investigate a subset of BioPortal for the question of how the size of modules effects the hardness of subsumption tests. We were able to show that the majority of subsumption tests triggered did not change at all in hardness from a sub- to its super-module. We concluded that OWL reasoners, especially Pellet and HermiT, were only to a very small degree sensitive to modularly irrelevant axioms. We were able to isolate a number of subsumption tests that were harder in the sub-module than in the super-module, but remained unsure about the *cause* of that effect. The high degree of perceived stochasticity in the classification process and the much higher potential for measurement error in the microsecond area precluded any conclusions about the pathological nature of

these tests. However, on average, reasoners appeared to profit, if slightly, from modularity in terms of subsumption test hardness. In the last chapter, we moved on to observe decomposition-based classification techniques in action. Not unexpectedly, decomposition-based classification techniques often have a great detrimental effect on classification performance, due to the occasionally severe degree of redundancy induced by overlapping modules or, to a lesser degree, the overhead induced by the inefficiency of current decomposition implementations. We were able to observe some strong positive effects in terms of search space pruning and average subsumption test hardness reduction. Some other cases where decomposition-based classification proved detrimental on the other hand led us to conclude that the applicability of a particular decomposition-based strategy depends heavily on the nature of the particular ontology at hand.

Many of the results in this thesis are unsatisfyingly negative, or at least not clearly positive. Our initial intention was to show *how* beneficial modules could be, and that we would find some clear answer of the type: "the smaller the module, the easier the test" or "the more (smaller) modules in the decomposition, the higher the effect of traversal space pruning". We have not arrived at that point. In fact, the doubt that we initially intended to dispel is still there, only some of the folklore like "Atomic Decomposition-based reasoning will never work" or "reasoning with a module is of course easier" have now been substantiated a bit more, and the observations we made form a sound basis for future work in this direction. These results should shed some light into the performance of modularity-based reasoners such as Chainsaw and MORe in practice. In particular, the factors that impact the time both negatively (e.g. overhead, redundancy) and positively (e.g. traversal space pruning) are isolated and their prevalence and magnitude observed empirically. Apart from increasing our understanding on modularity-based reasoning, we have gained some insight on classification in general, for example the average hardness of tests and test counts and the characterisation of ontologies that do not force any tests to be triggered—or particularly many.

Apart from generating insight, we have gathered a significant corpus of data about classification with modules[1] and learned a lot about experimental methodologies. It was not a focus of this thesis to understand classification performance as such better, nor to characterise the delegate reasoners themselves in depth

[1]The data collected as part of this thesis are available at the supplementary materials website.

with respect to their classification behaviour. However, we hope that the data we have generated will serve as a stepping stone for researchers that are interested in understanding classification better. A completely unexplored dimension of the dataset for example is the order of the tests triggered: if a relationship could be established between test order and the overall classification time, it may be possible to develop heuristics that force more beneficial test orders. Another community we expect to benefit from our dataset investigates (typically machine learning based) methods to predict classification time. Prediction models often depend on large amounts of training data, and as gathering such data is very expensive (both from a time and from an effort perspective), and as our dataset contains data from multiple runs, classification times and metadata of random subsets and modules and millions of individual subsumption test measurements, it should prove a valuable resource. Some of the methodological insights we gained were related to the variance across classification runs. Depending on the ontology and the reasoner, variance can be extremely high, an observation that should once and for all end the unjustified one-run policy still prevailing in our community. We have gathered some preliminary evidence that benchmarking in the cloud does not result in a higher degree of variance than using a (local) cluster of typical desktop machines. The high degree of fluctuations in test counts suggest that classification is in many cases subject to stochastic effects. Pinpointing these effects is a part of future work, but only being aware of them should again at least put another weight against the one-run practice of our community (in particular when verifying the performance of test avoiding optimisations through counting).

8.2 Outstanding Issues and Future Work

Many aspects of the research presented as part of this thesis are still in an early stage (stage benchmark, modular reasoning). We hope that we were able to lay the foundations for systematic investigations into the effect of modularity on reasoning hardness. However, many open questions, both technical and methodological, remain.

8.2.1 Modular Reasoning

With respect to modular reasoning, we want to address a number of open questions. At the time of writing, we are experimenting with dynamic dispatch to OWL 2 EL delegate reasoners, i.e. in the case where a chunk in the decomposition is in OWL 2 EL, automatically dispatch it to an appropriate delegate reasoner. We would like to extend this route of investigation to all sorts of specialised reasoners, and see whether this can make a difference. In this thesis, we referred to this problem as the *optimal dispatch* problem. The goal of this research is the development of heuristics that could guide the automated assignment of appropriate delegate reasoners to particular chunks in the decomposition.

Conceptually, we want to tackle the problem of communicating known non-subsumptions between delegate reasoners, preferably without having to manipulate the existing implementations too much. One way to achieve this is with ABox witnesses: If we assert an individual to be an instance of $A \sqcap \neg B$, then we know that A cannot be a subclass of B. However, the potentially huge number of necessary witnesses and the subsequent detrimental effects are likely to outweigh the burden of simply triggering the SAT test again. Therefore, we are looking for ways to improve the representation of non-entailments.

The next step in our analysis of modular techniques is to characterise in more depth the ontologies that are likely to profit from decomposition-based (or modular) reasoning. So far, we only have an intuition that ontologies that cause reasoners to do a lot of subsumption testing are likely to benefit the most. Based on those characterisations, heuristics could be developed to decide whether a particular decomposition is likely to give any benefits at all.

Lastly, we want to increase our understanding of the stochastic effects impacting classification in general. We are mainly interested in two questions: (1) What are the exact sources of stochasticity? These could be non-deterministic choices during a satisfiability check or a random effect induced by the programming environment (for example, random keys in a hash set). (2) If stochasticity has a significant effect on classification performance, can we develop heuristics based on our data that allow algorithms to make *better* choices, i.e. to force them into a more benign space?

8.2.2 Experimental Pipeline

My main interest with respect to OWL and reasoning is improving the experimental pipeline that helps with evaluations, in particular with respect to the following aspects:

- Datasets: Ontology corpora and experimental analyses.
- Experimental Setup: Infrastructure, guides and checklists.
- Reproducibility: Fully executable analysis pipelines, from data gathering all the way through analysis.

One of the starting points of our tour into the world of OWL reasoner benchmarking was the detailed characterisation of existing OWL corpora and the definition of new ones. It was striking to see that people were still using either old corpora such as Tones[2] or browsing the web and manually assembling corpora.[3] We have made some efforts to gather information on existing corpora[4] and make them available. However, a lot still needs to be done in terms of delivery. At the moment, we provide a service for ontology researchers to assemble suitable corpora for their problems (for example all RL ontologies larger than 100 axioms, or all ontologies with unsatifiable classes). At the moment we create these sets more or less manually by executing a series of scripts that filter, de-duplicate and characterise the ontologies according to the input criteria. In the future, we would like to fully automate the entire process: from the dataset definition by the client to the automated assembly and full characterisation of the corpus (metrics, plots, reports).

A second (related) aspect is the re-use of analysis results. We recently received a request for the results of the ORE reasoner competition to drive an experiment into using Machine Learning to predict OWL reasoner robustness. This dataset covers roughly 7.5 days worth of machine hours on classifications. Many months worth of classification time measurements were gathered as part of this thesis. It would be both a saving for the environment (think carbon footprint) and a reduction of human effort if datasets like this would be shared in a form that made them usable by others. We have started building an OWL experiment

[2] In fact, at the time of this writing, I received a request for a copy of Tones—September 2015.

[3] A colleague in the community recently "downloaded" BioPortal by manually navigating to all 350 or so entries and downloading the ontologies from their websites—one by one.

[4] `http://mowlrepo.cs.manchester.ac.uk/`

ontology[5] (mainly as a structured vocabulary for the metrics used as part of our experiments), and we would like to integrate this in the future with STATO and other efforts to share our experimental data.

Our next concern is the improvement of OWL experiment infrastructure. We have recently started running experiments in the cloud (Experiment 7.1). There is however as yet little knowledge about how to control the variance when experimenting with non-dedicated virtual servers, how many runs are necessary to ensure a meaningful classification time result and how to distinguish variance caused by algorithmic non-determinism from measurement error. We are producing some guidelines and checklists for the community to run experiments on the cloud, covering many of the typical problems OWL researchers have with experimentation (timeouts, OWL API, metrics gathering, etc.), but also plan to run some of our own experiments to investigate the above mentioned problems.

Lastly, we want to help the community to make experiments fully reproducible. We made some first steps by publishing datasets with DOI's on a data publishing platform (Zenodo[6]), and coding up our analyses completely in R. There are however still a lot of disconnected components to be tied together, for example the integration of the metrics gathering and analyses that are conducted as part of a Java-based experiment and the actual analysis scripts (Python, R, Matlab).

[5] `https://github.com/matentzn/owlexperimentontology`
[6] `https://zenodo.org/`

Bibliography

[AM05] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162(12):49–88, 2005. (Cited on pages 60 and 61.)

[AYL15] Nourhne Alaya, Sadok Ben Yahia, and Myriam Lamolle. What Makes Ontology Reasoning So Arduous?: Unveiling the Key Ontological Features. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, New York, NY, USA, 2015. (Cited on page 16.)

[BCM+07] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications (Second Edition)*. Cambridge University Press, Cambridge, 2007. (Cited on pages 10, 13, 25, 33, and 34.)

[BDGR12] Fernando Bobillo, Miguel Delgado, and Juan Gmez-Romero. DeLorean: A Reasoner for Fuzzy OWL 2. *Expert Systems with Applications*, 39(1):258–272, 2012. (Cited on page 231.)

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), 2008. (Cited on page 166.)

[BH08] Marian Babik and Ladislav Hluch. A Testing Framework for OWL-DL Reasoning. In *Fourth International Conference on Semantics, Knowledge and Grid, SKG '08, Beijing, China, December 3-5, 2008*, pages 42–48, 2008. (Cited on page 91.)

[BHJ09] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *Proceedings of the Third International Conference on Weblogs and Social Media, ICWSM 2009, San Jose, California, USA, May 17-20, 2009.*, 2009. (Cited on page 166.)

[BHN+94] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jrgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 4(2):109–132, 1994. (Cited on page 35.)

[BLS06] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL - A Polynomial-Time Reasoner for Life Science Ontologies. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 287–291, 2006. (Cited on page 231.)

[BNJ14] Jaroslaw Bak, Maciej Nowak, and Czeslaw Jedrzejek. RuQAR: Reasoning Framework for OWL 2 RL Ontologies. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pages 195–198, 2014. (Cited on page 231.)

[Boy08] Brent Boyer. Robust Java Benchmarking, Part 1: Issues. Technical report, IBM developerWorks, 2008. (Cited on page 87.)

[BP10] Franz Baader and Rafael Pealoza. Axiom Pinpointing in General Tableaux. *Journal of Logic and Compution*, 20(1):5–34, 2010. (Cited on page 26.)

[BPS11] Samantha Bail, Bijan Parsia, and Ulrike Sattler. Extracting Finite Sets of Entailments from OWL Ontologies. In *Informal Proceedings of the 24th International Workshop on Description Logics (DL-2011), Barcelona, Spain, July 13-16, 2011.*, 2011. (Cited on page 21.)

[BS08] Fernando Bobillo and Umberto Straccia. fuzzyDL: An Expressive Fuzzy Description Logic Reasoner. In *FUZZ-IEEE 2008, IEEE International Conference on Fuzzy Systems, Hong Kong, China, 1-6 June, 2008, Proceedings*, pages 923–930, 2008. (Cited on page 231.)

[BVSH09] Jie Bao, George Voutsadakis, Giora Slutzki, and Vasant Honavar. Package-Based Description Logics. In *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*,

pages 349–371. Springer-Verlag Berlin Heidelberg, 2009. (Cited on pages 39 and 62.)

[CGL+11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO System for Ontology-based Data Access. *Semantic Web*, 2(1):43–53, 2011. (Cited on page 231.)

[Con13] The Gene Ontology Consortium. Gene Ontology Annotations and Resources. *Nucleic Acids Research*, 41(D1):D530–D535, 2013. (Cited on page 10.)

[DCtTdK11] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. Comparison of Reasoners for Large Ontologies in the OWL 2 EL Profile. *Semantic Web*, 2(2):71–87, 2011. (Cited on pages 91 and 92.)

[DLZC13] Chan Le Duc, Myriam Lamolle, Antoine Zimmermann, and Olivier Cur. DRAOn: A Distributed Reasoner for Aligned Ontologies. In *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013.*, pages 81–86, 2013. (Cited on pages 62 and 231.)

[DV13] Chiara Del Vescovo. *The Modular Structure of an Ontology: Atomic Decomposition and its Applications.* PhD thesis, University of Manchester, 2013. (Cited on pages 35, 36, 37, 40, 41, 42, 43, 70, and 164.)

[DVPSS11a] Chiara Del Vescovo, Bijan Parsia, Ulrike Sattler, and Thomas Schneider. The Modular Structure of an Ontology: Atomic Decomposition. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2232–2237, 2011. (Cited on pages 39 and 42.)

[DVPSS11b] Chiara Del Vescovo, Bijan Parsia, Ulrike Sattler, and Thomas Schneider. The Modular Structure of an Ontology: Atomic Decomposition and Module Count. In *Modular Ontologies - Proceedings of*

the Fifth International Workshop, WoMO 2011, Ljubljana, Slovenia, August 2011, pages 25–39, 2011. (Cited on pages 106 and 107.)

[EOS+12] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query Rewriting for Horn-SHIQ Plus Rules. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-2012), Toronto, Ontario, Canada, July 22-26, 2012.*, 2012. (Cited on page 231.)

[For10] Santo Fortunato. Community Detection in Graphs. *Physics Reports*, 486(3):75–174, 2010. (Cited on page 166.)

[GBE07] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically Rigorous Java Performance Evaluation. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*, pages 57–76, 2007. (Cited on page 149.)

[GBJR+13] Rafael S. Gonalves, Samantha Bail, Ernesto Jimnez-Ruiz, Nicolas Matentzoglu, Bijan Parsia, Birte Glimm, and Yevgeny Kazakov. OWL Reasoner Evaluation (ORE) Workshop 2013 Results: Short Report. In *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013.*, pages 1–18, 2013. (Cited on pages 14, 20, 90, and 91.)

[GCH+13] Martin Giese, Diego Calvanese, Peter Haase, Ian Horrocks, Yannis Ioannidis, Herald Kllapi, Manolis Koubarakis, Maurizio Lenzerini, Ralf Mller, and others. Scalable End-user Access to Big Data. *Big Data Computing*, pages 205–245, 2013. (Cited on page 90.)

[GHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular Reuse of Ontologies: Theory and Practice. *Journal of Artificial Intelligence Research*, 31:273–318, 2008. (Cited on pages 13, 14, 26, 35, and 36.)

[GHM+08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The Next Step

for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008. (Cited on pages 11 and 22.)

[GHM+12] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A Novel Approach to Ontology Classification. *Journal of Web Semantics*, 14:84–101, 2012. (Cited on pages 16, 28, 35, and 92.)

[GHM+14] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014. (Cited on pages 49, 112, 152, and 231.)

[GHWK07] Bernardo Cuenca Grau, Christian Halaschek-Wiener, and Yevgeny Kazakov. History Matters: Incremental Ontology Reasoning Using Modules. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, pages 183–196, 2007. (Cited on pages 13, 58, and 59.)

[GI13] Andrey V. Grigorev and Alexander G. Ivashko. TReasoner: System Description. In *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013.*, pages 26–31, 2013. (Cited on page 231.)

[GKW14] William Gatens, Boris Konev, and Frank Wolter. Lower and Upper Approximations for Depleting Modules of Description Logic Ontologies. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 345–350, 2014. (Cited on page 38.)

[GM12] Mara del Mar Roldn Garca and Jos Francisco Aldana Montes. Evaluating DBOWL: A Non-materializing OWL Reasoner based on Relational Database Technology. In *Informal Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1, 2012.*, 2012. (Cited on page 231.)

[GMPS13] Rafael S. Gonalves, Nicolas Matentzoglu, Bijan Parsia, and Uli Sattler. The Empirical Robustness of Description Logic Classification.

In *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013*, pages 277–280, 2013. (Cited on pages 11, 12, 16, 20, 91, and 92.)

[Gon14] Rafael S. Gonalves. *Impact Analysis in Description Logic Ontologies*. PhD thesis, University of Manchester, 2014. (Cited on page 229.)

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005. (Cited on page 91.)

[GPS12] Rafael S. Gonalves, Bijan Parsia, and Ulrike Sattler. Performance Heterogeneity and Approximate Reasoning in Description Logic Ontologies. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pages 82–98, 2012. (Cited on pages 52, 57, 62, 71, 101, 102, 107, 108, 111, 132, and 142.)

[GTH06] Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. Framework for an Automated Comparison of Description Logic Reasoners. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, pages 654–667, 2006. (Cited on page 91.)

[HB11] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1):11–21, 2011. (Cited on pages 26 and 77.)

[HHMW12] Volker Haarslev, Kay Hidde, Ralf Mller, and Michael Wessel. The RacerPro Knowledge Representation and Reasoning System. *Semantic Web*, 3(3):267–277, 2012. (Cited on page 231.)

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible SROIQ. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 57–67, 2006. (Cited on pages 11, 12, 22, 25, and 71.)

[Hla05] Jan Hladik. A Generator for Description Logic Formulas. In *Informal Proceedings of the 18th International Workshop on Description Logics (DL-2005), Edinburgh, Scotland, UK, July 26-28, 2005.*, 2005. (Cited on page 15.)

[HM01] Volker Haarslev and Ralf Mller. RACER System Description. In *First International Joint Conference of Automated Reasoning (IJCAR-2001), Siena, Italy, June 18-23, 2001.*, pages 701–706, 2001. (Cited on pages 12 and 15.)

[HMP+14] Matthew Horridge, Jonathan Mortensen, Bijan Parsia, Ulrike Sattler, and Mark A. Musen. A Study on the Atomic Decomposition of Ontologies. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, pages 65–80, 2014. (Cited on page 71.)

[HMT01] Volker Haarslev, Ralf Mller, and Anni-Yasmin Turhan. Exploiting Pseudo Models for TBox and ABox Reasoning in Expressive Description Logics. In *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, pages 61–75, 2001. (Cited on page 127.)

[Hom07] Martin Homola. Distributed Description Logics Revisited. In *Informal Proceedings of the 20th International Workshop on Description Logics (DL-2007), Brixen-Bressanone, Italy, 8-10 June, 2007.*, 2007. (Cited on page 62.)

[Hor97] Ian R Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997. (Cited on pages 12 and 158.)

[Hor98] Ian Horrocks. The FaCT System. In *Proceedings of the International Conference of Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-98), Oisterwijk, The Netherlands, May 5-8, 1998.*, pages 307–312, 1998. (Cited on pages 12 and 15.)

[Hor11] Matthew Horridge. *Justification-based Explanation in Ontologies*. PhD thesis, University of Manchester, 2011. (Cited on page 26.)

[HPS98] Ian Horrocks and Peter F. Patel-Schneider. DL Systems Comparison (Summary Relation). In *Informal Proceedings of the 11th International Workshop on Description Logics (DL-1998), Povo-Trento, Italy, June 6-8, 1998.*, 1998. (Cited on page 91.)

[Kaz08] Yevgeny Kazakov. RIQ and SROIQ Are Harder than SHOIQ. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-2008), Sydney, Australia, September 16-19, 2008.*, pages 274–284, 2008. (Cited on page 12.)

[Kaz09] Yevgeny Kazakov. Consequence-Driven Reasoning for Horn SHIQ Ontologies. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009), Pasadena, California, USA, July 11-17, 2009.*, pages 2040–2045, 2009. (Cited on pages 33 and 71.)

[KJM+12] Julie Klein, Simon Jupp, Panagiotis Moulos, Myriem Fernandez, Bndicte Buffin-Meyer, Audrey Casemayou, Rana Chaaya, Aristidis Charonis, Jean-Loup Bascands, Robert Stevens, and others. The KUPKB: a Novel Web Application to Access Multiomics Data on Kidney Disease. *The FASEB Journal*, 26(5):2145–2153, 2012. (Cited on page 11.)

[KK13] Yevgeny Kazakov and Pavel Klinov. Incremental Reasoning in OWL EL without Bookkeeping. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 232–247, 2013. (Cited on page 59.)

[KKS14] Yevgeny Kazakov, Markus Krtzsch, and Frantisek Simancik. The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with EL Ontologies. *Journal of Automated Reasoning*, 53(1):1–61, 2014. (Cited on pages 12, 25, 28, 33, 44, 53, and 231.)

[KLK12] Yong-Bin Kang, Yuan-Fang Li, and Shonali Krishnaswamy. Predicting Reasoning Performance Using Ontology Metrics. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web*

Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I, pages 198–214, 2012. (Cited on page 16.)

[KLPW10] Boris Konev, Carsten Lutz, Denis Ponomaryov, and Frank Wolter. Decomposing Description Logic Ontologies. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR-2010), Toronto, Ontario, Canada, May 9-13, 2010.*, 2010. (Cited on page 39.)

[KRRM+14] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pages 552–567, 2014. (Cited on page 231.)

[LMPS15] Michael Lee, Nicolas Matentzoglu, Bijan Parsia, and Uli Sattler. A multi-reasoner, justification-based approach to reasoner correctness. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, pages 393–408, 2015. (Cited on pages 20, 25, 82, 146, and 182.)

[LWW07] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative Extensions in Expressive Description Logics. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 453–458, 2007. (Cited on page 36.)

[MBK06] Christopher J. Matheus, Kenneth Baclawski, and Mieczyslaw M. Kokar. BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML and R-Entailment Rules. In *Rules and Rule Markup Languages for the Semantic Web, Second International Conference, RuleML 2006, Athens, Georgia, USA, November 10-11, 2006, Proceedings*, pages 67–74, 2006. (Cited on page 231.)

[MBP13a] Nicolas Matentzoglu, Samantha Bail, and Bijan Parsia. A Corpus of

OWL DL Ontologies. In *Informal Proceedings of the 26th International Workshop on Description Logics (DL-2013), Ulm, Germany, July 23 - 26, 2013.*, pages 829–841, 2013. (Cited on page 19.)

[MBP13b] Nicolas Matentzoglu, Samantha Bail, and Bijan Parsia. A Snapshot of the OWL Web. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 331–346, 2013. (Cited on pages 19 and 93.)

[MDOS14] Chris Mungall, Heiko Dietze, and David Osumi-Sutherland. Use of OWL within the Gene Ontology. In *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014) co-located with 13th International Semantic Web Conference on (ISWC 2014), Riva del Garda, Italy, October 17-18, 2014.*, pages 25–36, 2014. (Cited on page 11.)

[Men12] Julian Mendez. jcel: A Modular Rule-based Reasoner. In *Informal Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1, 2012.*, 2012. (Cited on page 231.)

[MGH+12] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles (Second Edition), December 2012. (Cited on pages 23 and 94.)

[MHM13] Raghava Mutharaju, Pascal Hitzler, and Prabhaker Mateti. DistEL: A Distributed EL+ Ontology Classifier. In *Proceedings of the 9th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS-2015), Sydney, Australia, October 21, 2013.*, pages 17–32, 2013. (Cited on page 33.)

[MHML15] Raghava Mutharaju, Pascal Hitzler, Prabhaker Mateti, and Freddy Lcu. Distributed and Scalable OWL EL Reasoning. In *Proceedings of the 12th European Semantic Web Conference, (ESWC-2015), Portoroz, Slovenia, May 31 - June 4, 2015.*, pages 88–103, 2015. (Cited on page 231.)

[MJL13] Alejandro Metke-Jimenez and Michael Lawley. Snorocket 2.0: Concrete Domains and Concurrent Classification. In *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013.*, pages 32–38, 2013. (Cited on pages 33 and 231.)

[MKC14] Till Mossakowski, Oliver Kutz, and Mihai Codescu. Ontohub: A Semantic Repository for Heterogeneous Ontologies. In *Proceedings of the Theory Day in Computer Science (DACS-2014), Bucharest, Romania, 2014.*, 2014. (Cited on page 10.)

[MLH+15] Nicolas Matentzoglu, Jared Leo, Valentino Hudhra, Uli Sattler, and Bijan Parsia. A Survey of Current, Stand-alone OWL Reasoners. In *Informal Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015), Athens, Greece, June 6, 2015.*, pages 68–79, 2015. (Cited on pages 19, 26, 51, 59, 68, 97, and 230.)

[MNP+14] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence (AAAI-2014), Qubec City, Qubec, Canada, July 27 -31, 2014.*, pages 129–137, 2014. (Cited on page 231.)

[MP14a] Nicolas Matentzoglu and Bijan Parsia. *The Manchester OWL Corpus (MOWLCorp),Original Serialisation*. University of Manchester, July 2014. (Cited on page 20.)

[MP14b] Nicolas Matentzoglu and Bijan Parsia. The OWL Full/DL Gap in the Field. In *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014) co-located with 13th International Semantic Web Conference on (ISWC 2014), Riva del Garda, Italy, October 17-18, 2014.*, pages 49–60, 2014. (Cited on pages 19 and 94.)

[MP14c] Nicolas Matentzoglu and Bijan Parsia. OWL/ZIP: Distributing Large and Modular Ontologies. In *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED*

2014) co-located with 13th International Semantic Web Conference on (ISWC 2014), Riva del Garda, Italy, October 17-18, 2014., pages 37–48, 2014. (Cited on pages 19, 71, and 118.)

[MPS14] Nicolas Matentzoglu, Bijan Parsia, and Uli Sattler. An Empirical Investigation of Difficulty of Subsets of Description Logic Ontologies. In *Informal Proceedings of the 27th International Workshop on Description Logics (DL-2014), Vienna, Austria, July 17-20, 2014.*, pages 659–670, 2014. (Cited on page 19.)

[MRW14] Francisco Martn-Recuerda and Dirk Walther. Fast Modularisation and Atomic Decomposition of Ontologies Using Axiom Dependency Hypergraphs. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, pages 49–64, 2014. (Cited on page 43.)

[MRW15] Francisco Martin-Recuerda and Dirk Walther. HyS: Fast Atomic Decomposition and Module Extraction of OWL-EL Ontologies. *Semantic Web, Unpublished, Major Revision*, 2015. (Cited on page 43.)

[MSP15] Nicolas Matentzoglu, Uli Sattler, and Bijan Parsia. Empirical Investigation of Subsumption Test Hardness in Description Logic Classification. In *Informal Proceedings of the 28th International Workshop on Description Logics (DL-2015), Athens, Greece, 7-10, 2015.*, 2015. (Cited on page 19.)

[MTPS14] Nicolas Matentzoglu, Daniel Tang, Bijan Parsia, and Uli Sattler. The Manchester OWL Repository: System Description. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, pages 285–288, 2014. (Cited on page 93.)

[MYQ+06] Li Ma, Yang Yang, Zhaoming Qiu, Guo Tong Xie, Yue Pan, and Shengping Liu. Towards a Complete OWL Ontology Benchmark. In *The Semantic Web: Research and Applications, 3rd European*

Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings, pages 125–139, 2006. (Cited on page 91.)

[NNS11] Mathias Niepert, Jan Noessner, and Heiner Stuckenschmidt. Log-Linear Description Logics. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2153–2158, 2011. (Cited on page 231.)

[NSW+09] Natalya Fridman Noy, Nigam H. Shah, Patricia L. Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L. Rubin, Margaret-Anne D. Storey, Christopher G. Chute, and Mark A. Musen. BioPortal: Ontologies and Integrated Data Resources at the Click of a Mouse. *Nucleic Acids Research*, 37(Web-Server-Issue):170–173, 2009. (Cited on pages 10, 92, and 93.)

[Pal15] Ignazio Palmisano. JFact Reasoner Repository, 2015. (Cited on page 231.)

[PCMB15] Niels Peek, Carlo Combi, Roque Marin, and Riccardo Bellazzi. Thirty Years of Artificial Intelligence in Medicine (AIME) Conferences: A Review of Research Themes. *Artificial Intelligence in Medicine*, 2015. (Cited on page 10.)

[PGSH14] Sambhawa Priya, Yuanbo Guo, Michael Spear, and Jeff Heflin. Partitioning OWL Knowledge Bases for Parallel Reasoning. In *Proceedings of the International Conference on Semantic Computing, Newport Beach, CA, USA, June 16-18, 2014.*, pages 108–115, 2014. (Cited on page 61.)

[PLTS07] Thi Anh Le Pham, Nhan Le-Thanh, and Peter Sander. Some Approaches of Ontology Decomposition in Description Logics. In *Complex Systems Concurrent Engineering*, pages 537–546. Springer London, 2007. (Cited on page 61.)

[PLTS08] Thi Anh Le Pham, Nhan Le-Thanh, and Peter Sander. Decomposition-based Reasoning for Large Knowledge Bases in Description Logics. *Integrated Computer-Aided Engineering*, 15(1):53–70, 2008. (Cited on page 61.)

[PMG+15] Bijan Parsia, Nicolas Matentzoglu, Rafael S. Gonalves, Birte Glimm, and Andreas Steigmiller. The OWL Reasoner Evaluation (ORE) 2015 Competition Report (accepted). In *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS-2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, 2015. (Cited on pages 12, 14, 17, 60, 68, 69, 90, 93, and 96.)

[PS00] Peter F. Patel-Schneider. System Description: DLP. In *Proceedings of the 17th International Conference on Automated Deduction (CADE-17), Pittsburgh, PA, USA, June 17-20, 2000, Proceedings.*, pages 297–301, 2000. (Cited on pages 12 and 15.)

[PS10] Bijan Parsia and Thomas Schneider. The Modular Structure of an Ontology: An Empirical Study. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*, 2010. (Cited on page 106.)

[R C15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. (Cited on page 77.)

[RAK06] Rachel L. Richesson, James E. Andrews, and Jeffrey P. Krischer. Use of SNOMED CT to Represent Clinical Research Data: A Semantic Characterization of Data Items on Case Report Forms in Vasculitis Research. *Journal of the American Medical Informatics Association*, 13(5):536–546, 2006. (Cited on page 10.)

[RBLZ13] Fabrizio Riguzzi, Elena Bellodi, Evelina Lamma, and Riccardo Zese. BUNDLE: A Reasoner for Probabilistic Ontologies. In *Proceedings of the 7th International Conference of Web Reasoning and Rule Systems (RR-2013), Mannheim, Germany, July 27-29, 2013.*, pages 183–197, 2013. (Cited on page 231.)

[RGH12] Ana Armas Romero, Bernardo Cuenca Grau, and Ian Horrocks. MORe: Modular Combination of OWL Reasoners for Ontology

Classification. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pages 1–16, 2012. (Cited on pages 13, 14, 43, 52, 71, 77, 82, and 231.)

[RKGH15] Ana Armas Romero, Mark Kaminski, Bernardo Cuenca Grau, and Ian Horrocks. Ontology Module Extraction via Datalog Reasoning. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI-2015), January 25-30, 2015, Austin, Texas, USA.*, pages 1410–1416, 2015. (Cited on pages 37 and 71.)

[SAMN13] Manuel Salvadores, Paul R. Alexander, Mark A. Musen, and Natalya Fridman Noy. BioPortal as a Dataset of Linked Biomedical Ontologies and Terminologies in RDF. *Semantic Web*, 4(3):277–284, 2013. (Cited on page 93.)

[SCVJ08] Forrest J. Shull, Jeffrey C. Carver, Sira Vegas, and Natalia Juristo. The Role of Replications in Empirical Software Engineering. *Empirical Software Engineering*, 13(2):211–218, 2008. (Cited on page 90.)

[Ser13] Baris Sertkaya. The ELepHant Reasoner System Description. In *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013.*, pages 87–93, 2013. (Cited on page 231.)

[SGL14] Andreas Steigmiller, Birte Glimm, and Thorsten Liebig. Coupling Tableau Algorithms for Expressive Description Logics with Completion-Based Saturation Procedures. In *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, pages 449–463, 2014. (Cited on page 29.)

[SLG14] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System Description. *Journal of Web Semantics*, 27:78–85, 2014. (Cited on pages 12, 29, 68, 92, 97, and 231.)

[SM15] Uli Sattler and Nicolas Matentzoglu. List of Reasoners (owl.cs), 2015. Modified: 01/09/2014. (Cited on page 97.)

[SMH14] Frantisek Simancik, Boris Motik, and Ian Horrocks. Consequence-based and Fixed-parameter Tractable Reasoning in Description Logics. *Artificial Intelligence*, 209:29–77, 2014. (Cited on page 15.)

[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):51–53, 2007. (Cited on pages 59 and 231.)

[SS11] Michael Schneider and Geoff Sutcliffe. Reasoning in the OWL 2 Full Ontology Language Using First-Order Automated Theorem Proving. In *Proceedings of 23rd International Conference on Automated Deduction (CADE-2011), Wroclaw, Poland, July 31 - August 5, 2011.*, pages 461–475, 2011. (Cited on page 23.)

[SSB14] Viachaslau Sazonau, Uli Sattler, and Gavin Brown. Predicting Performance of OWL Reasoners: Locally or Globally? In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR-2014), Vienna, Austria, July 20-24, 2014.*, 2014. (Cited on page 16.)

[SSD12] Weihong Song, Bruce Spencer, and Weichang Du. WSReasoner: A Prototype Hybrid Reasoner for ALCHOI Ontology Classification using a Weakening and Strengthening Approach. In *Informal Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1, 2012.*, 2012. (Cited on page 231.)

[SSZ09] Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which Kind of Module Should I Extract? In *Informal Proceedings of the 22nd International Workshop on Description Logics (DL-2009), Oxford, UK, July 27-30, 2009.*, 2009. (Cited on pages 36 and 132.)

[ST05] Luciano Serafini and Andrei Tamilin. DRAGO: Distributed Reasoning Architecture for the Semantic Web. In *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, pages 361–376, 2005. (Cited on page 62.)

[TDKM14] Dorothea Tsatsou, Stamatia Dasiopoulou, Ioannis Kompatsiaris, and Vasileios Mezaris. LiFR: A Lightweight Fuzzy DL Reasoner. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pages 263–267, 2014. (Cited on page 231.)

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 292–297, 2006. (Cited on pages 25, 35, 66, 90, and 231.)

[TNNM13] Tania Tudorache, Csongor Nyulas, Natalya Fridman Noy, and Mark A. Musen. Using Semantic Web in ICD-11: Three Years Down the Road. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, pages 195–211, 2013. (Cited on page 10.)

[Tob01] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, Technische Universitt Dresden, 2001. (Cited on page 12.)

[TP12a] Dmitry Tsarkov and Ignazio Palmisano. Chainsaw: a Metareasoner for Large Ontologies. In *Informal Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1, 2012.*, 2012. (Cited on pages 13, 42, 52, 54, 56, 77, 80, and 231.)

[TP12b] Dmitry Tsarkov and Ignazio Palmisano. Divide et Impera: Metareasoning for Large Ontologies. In *Informal Proceedings of the 9th Workshop on OWL: Experiences and Directions (OWLED-2012), Heraklion, Crete, Greece, May 27-28, 2012.*, 2012. (Cited on page 54.)

[TPR10] Edward Thomas, Jeff Z. Pan, and Yuan Ren. TrOWL: Tractable OWL 2 Reasoning Infrastructure. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II*, pages 431–435, 2010. (Cited on pages 25 and 231.)

[Tsa12] Dmitry Tsarkov. Improved Algorithms for Module Extraction and Atomic Decomposition. In *Informal Proceedings of the 25th International Workshop on Description Logics (DL-2012), Rome, Italy, June 7-10, 2012.*, 2012. (Cited on page 43.)

[Tsa14] Dmitry Tsarkov. Incremental and Persistent Reasoning in FaCT++. In *Informal Proceedings of the 3rd International Workshop on OWL Reasoner Evaluation (ORE 2014), Vienna, Austria, July 13, 2014.*, pages 16–22, 2014. (Cited on page 59.)

[TV03] Christoph Tempich and Raphael Volz. Towards a Benchmark for Semantic Web Reasoners - An Analysis of the DAML Ontology Library. In *EON2003, Evaluation of Ontology-based Tools, Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools held at the 2nd International Semantic Web Conference ISWC 2003, 20th October 2003 (Workshop day), Sundial Resort, Sanibel Island, Florida, USA*, 2003. (Cited on page 91.)

[VKP+13] Chiara Del Vescovo, Pavel Klinov, Bijan Parsia, Ulrike Sattler, Thomas Schneider, and Dmitry Tsarkov. Empirical Study of Logic-Based Modules: Cheap Is Cheerful. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 84–100, 2013. (Cited on pages 36, 42, and 67.)

[VNP15] Edgaras Valincius, Hai H. Nguyen, and Jeff Z. Pan. A Power Consumption Benchmark Framework for Ontology Reasoning on Android Devices. In *Informal Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015), Athens, Greece, June 6, 2015.*, pages 80–86, 2015. (Cited on page 16.)

[WH14] Kejia Wu and Volker Haarslev. Parallel OWL Reasoning: Merge Classification. In *Revised Selected Papers of the Third Joint International Conference of Semantic Technology (JIST-2013), Seoul, South Korea, November 28-30, 2013.*, pages 211–227, 2014. (Cited on page 61.)

[WLL+07] Timo Weithner, Thorsten Liebig, Marko Luther, Sebastian Bhm, Friedrich W. von Henke, and Olaf Noppens. Real-World Reasoning

with OWL. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, pages 296–310, 2007. (Cited on page 91.)

[WPH06] Taowei David Wang, Bijan Parsia, and James A. Hendler. A Survey of the Web Ontology Landscape. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, pages 682–694, 2006. (Cited on page 27.)

[XE11] Guohui Xiao and Thomas Eiter. Inline Evaluation of Hybrid Knowledge Bases. In *Proceedings of the 5th International Conference of Web Reasoning and Rule Systems (RR-2011), Galway, Ireland, August 29-30, 2011.*, pages 300–305, 2011. (Cited on page 231.)

[ZBL+14] Riccardo Zese, Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi, and Fabiano Aguiari. Semantics and Inference for Probabilistic Description Logics. In *Uncertainty Reasoning for the Semantic Web III - ISWC International Workshops, URSW 2011-2013, Revised Selected Papers*, pages 79–99, 2014. (Cited on page 231.)

[ZBLR13] Riccardo Zese, Elena Bellodi, Evelina Lamma, and Fabrizio Riguzzi. A Description Logics Tableau Reasoner in Prolog. In *Proceedings of the 28th Italian Conference on Computational Logic, Catania, Italy, September 25-27, 2013.*, pages 33–47, 2013. (Cited on page 231.)

Appendix A

Appendix

A.1 List of ontologies in BioPortal Snapshot

$\mathcal{O}$	PROF	ABox	TBox	Expressivity	K1	K2
AAO	Profiled	0	2673	ALE		
ABA-AMB	Profiled	0	3441	ALCI		
ACGT-MO	OWL Full	96	5317	SROIQ(D)		
ADO	Pure DL	0	2400	SHI		X
ADW	OWL Full	3	635	ALCOF(D)		
AEO	Profiled	0	368	ALE		
AERO	Pure DL	81	1578	SHOIQ(D)		
AMINO-ACID	Pure DL	0	477	ALCF(D)		
APO	Profiled	0	304	AL		
ASDPTO	Profiled	0	283	AL		
ATC	Profiled	0	10153	AL		
ATMO	Profiled	0	208	ALE		
ATO	Profiled	0	12163	ALE		
AURA	Pure DL	714	37661	SHOIQ(D)		
BAO	Pure DL	0	4282	SROIQ(D)		X
BAO-GPCR	Pure DL	0	1010	ALERI+(D)		
BCO	Pure DL	25	365	SROIF(D)		X
BCTEO	Profiled	0	458	ALE		
BDO	Pure DL	26146	19806	SHIF(D)		X
BFO	Pure DL	0	95	ALC		
BHN	Profiled	0	2542	AL		
BHO	Pure DL	0	1925	ALCIF(D)		X
BIOMODELS	Pure DL	220948	439208	SRIF	X	
BIRNLEX	Profiled	0	3572	AL		
BMO	Profiled	0	336	ALE		
BMT	Profiled	0	282	ALI		
BNO	Profiled	0	104	AL		
BOF	Pure DL	768	603	ALCF(D)		
BP	Pure DL	0	381	SHIN(D)		
BRIDG	OWL Full	496	2808	SROIN(D)		

BRO	OWL Full	32	587	SHIF(D)		
BSAO	Profiled	0	353	ALE+		
BSPO	Pure DL	0	242	ALEHI+		
BT	Pure DL	0	837	SRI	X	
BTO	Profiled	0	6726	ALE		
CABRO	Pure DL	4	94	ALCHIQ		
CANCO	OWL Full	12	352	ALCH(D)		
CANONT	Profiled	0	51	AL		
CAO	Pure DL	0	428	SHIQ(D)	X	X
CARELEX	Profiled	0	321	ALH(D)		
CARO	Profiled	0	54	ALE+		
CARRE	Pure DL	0	454	ALCQ(D)		X
CBO	Profiled	0	342	SRF(D)		
CCO	Pure DL	0	2249883	ALEHI+		
CCON	Pure DL	364	214	SHOI		
CCONT	Pure DL	2	25112	SROIF		
CDAO	OWL Full	37	362	SROIQ(D)		
CHD	Profiled	0	508	AL		
CHEBI	Profiled	0	140957	ALE+		
CHEMBIO	Pure DL	0	475	SHIN(D)		
CHEMINF	OWL Full	71	1275	SROIN(D)		
CHMO	Profiled	0	3215	ALCH		
CL	Pure DL	0	18010	SH		
CLO	OWL Full	82541	64	SHIN(D)		
CMO	Profiled	0	3075	ALE+		
CMPO	Profiled	0	393	ALER+		
CNO	OWL Full	49	590	SHOIF(D)		
CO-WHEAT	Profiled	0	175	AL		
COGAT	Pure DL	0	4100	ALC		
COGPO	Pure DL	58	595	SHOIN(D)		X
CPRO	Pure DL	63	343	SHIF(D)		
CPTAC	Profiled	0	1118	ALCF(D)		
CSEO	Pure DL	0	26540	SRI(D)		
CSSO	Profiled	0	297	AL		
CTCAE	Profiled	0	6940	AL(D)		
CTX	Pure DL	1553	364	ALCOIN(D)		
DCM	OWL Full	3174	27	SHOIF(D)		
DCO-DEBUGIT	Pure DL	0	2531	SRIN(D)	X	
DDANAT	Profiled	0	347	ALE+		
DDI	Pure DL	50	339	SHOIN(D)		X
DERMLEX	OWL Full	18301	6151	ALUF(D)		
DERMO	Profiled	0	3992	ALE+		

DIAGONT	OWL Full	7	245	ALCOF(D)		
DIKB	Pure DL	12	643	ALCHOIN(D)		X
DOCCC	Profiled	177	203	AL(D)		
DPO	Profiled	0	2496	ALE		
DRON	Pure DL	21	747701	SHO	X	
ECG	Pure DL	0	1294	ALCIF(D)		X
ECO	Pure DL	0	1243	ALC		X
EDAM	OWL Full	0	4376	ALCI		
EDDA	Profiled	0	213	ALEI		
EHDA	Profiled	0	8339	ALE		
EHDAA	Profiled	0	2336	ALE		
EHDAA2	Profiled	0	13511	ALE+		
ELIG	Profiled	2	38	AL		
EMAP	Profiled	0	13730	ALE		
EMO	Pure DL	132	248	ALCRQ		X
ENVO	Pure DL	0	1970	SHI		
EOL	Profiled	0	659	ALE		
EP	OWL Full	48064	124618	SHF(D)		
EPILONT	Profiled	0	137	ALH(D)		
ERO	OWL Full	24	4964	SHOIF(D)		
FAO	Pure DL	0	115	ALEI+		
FB-BT	Profiled	0	24309	SH		
FB-CV	Profiled	0	896	SH		
FB-DV	Profiled	0	621	ALEH+		
FB-SP	Profiled	0	6587	AL		
FBBI	Profiled	0	548	S		
FHHO	Pure DL	12	512	ALCHIF(D)		X
FIRE	Profiled	16	105	ALCHOI		
FIX	Profiled	0	1684	ALE		
FLOPO	Profiled	0	31902	AL		
FMA	OWL Full	1076839	87056	ALCOIN(D)		
FYPO	Profiled	0	22854	SH		
GALEN	Pure DL	0	36738	ALEHIF+		
GAZ	Profiled	0	652361	ALE+		
GCO	Pure DL	0	10	ALIF		
GENE-CDS	Pure DL	0	4322	ALCQ		
GEXO	Pure DL	0	549075	SRI		
GFO	Pure DL	0	190	SHIQ		X
GFO-BIO	OWL Full	2	428	SHIN		
GFVO	Pure DL	0	113	ALCH(D)		
GLYCO	OWL Full	32887	595	SHOIQ(D)		
GMM	Profiled	0	1895	AL		

GO	Profiled	0	80762	ALEH+		
GO-EXT	Profiled	0	68775	SH		
GRO	Pure DL	7	945	ALCHIQ(D)		X
GRO-CPD	Profiled	0	235	ALE		
GRO-CPGA	Profiled	0	2434	S		
HAO	Profiled	0	4384	SR		
HEIO	Pure DL	11575	272	ALCHIF(D)		X
HINO	OWL Full	0	137761	ALERI+		
HIV	Profiled	0	9	AL+		
HIVO004	OWL Full	260	778	ALCHIQ(D)		
HL7	Profiled	0	8072	AL		
HOM	Profiled	0	83	ALC		
HP	Profiled	0	13832	AL		
HPIO	Pure DL	27	359	SHI		X
HRDO	Profiled	0	85019	ALEH		
HUGO	Profiled	0	32917	AL		
HUPSON	Pure DL	25	3704	SHOIF(D)		X
IAO	OWL Full	172	280	ALRIF+(D)		
ICD11-BODYSYSTEM	Profiled	0	28	AL		
ICECI	OWL Full	11376	2236	AL(D)		
ICF	OWL Full	17223	1991	ALCHOIF(D)		
ICO	Pure DL	16	715	SROIN(D)	X	
ICPS	OWL Full	4090	948	SHOIQ(D)		
IDO	Pure DL	83	1009	SROIF		
IDOBRU	Pure DL	63	7539	SROIQ(D)		
IDODEN	Pure DL	0	5958	SRIF		
IDOMAL	Profiled	0	3563	ALER+		
IDQA	Profiled	0	275	ALEI		
IFAR	Profiled	0	4977	AL		
IMGT-ONTOLOGY	OWL Full	0	14298	SHIN(D)		
IMMDIS	Profiled	0	1676	AL		
INO	OWL Full	0	514	ALERI+		
INTERNANO	Profiled	0	268	AL		
IXNO	Profiled	0	39	AL		
JERM	Pure DL	78	383	SHI(D)		X
KISAO	Pure DL	0	736	ALCRIQ(D)		X
LDA	Profiled	0	35	AL		
LHN	Profiled	0	347	ALE		
LIPRO	Pure DL	0	2349	ALCHIN		
LSM	Profiled	0	472	AL+		
MA	Profiled	0	4108	ALE+		
MAMO	Pure DL	0	136	ALCR		

MAT	Profiled	0	504	ALE		
MCBCC	Profiled	0	2732	ALCH(D)		
MCCL	Profiled	0	13584	ALCH(D)		
MCCV	OWL Full	0	99	ALUHIN(D)		
MEDO	Profiled	0	84	ALH		
MEGO	Profiled	0	431	ALE+		
MEO	Profiled	0	783	AL		
MF	Pure DL	20	1022	SROIQ		
MFO	Profiled	0	4402	ALE		
MFOEM	Pure DL	20	1311	SROIQ	X	
MHC	Pure DL	0	13781	ALCIQ(D)		X
MIRNAO	Profiled	0	764	ALEI		
MIRO	Profiled	0	4457	ALE+		
MIXS	Profiled	0	0	ALH		
MIXSCV	Profiled	0	518	AL		
MMO	Profiled	0	629	AL		
MO	OWL Full	932	494	ALEOF(D)		
MP	Profiled	0	13077	AL		
MPATH	Profiled	0	946	ALE+		
MS	Profiled	0	2878	ALE+		
NATPRO	OWL Full	91351	68647	SHOIN(D)		
NBO	Profiled	0	1296	ALE		
NCBITAXON	Profiled	0	847755	AL		
NCCO	Profiled	0	397	AL		
NCIT	Pure DL	89292	156923	SH(D)		
NEMO	Pure DL	182	2628	SHIQ(D)		
NEOMARK3	OWL Full	0	1213	ALCHQ(D)		
NEOMARK4	Pure DL	0	387	SHIQ		X
NGSONTO	Pure DL	0	244	SRQ(D)		X
NIFCELL	OWL Full	111	3433	SROIF		
NIFDYS	OWL Full	111	3387	SROIF		
NIFSUBCELL	OWL Full	111	3924	SROIF		
NIGO	Profiled	0	8833	SH		
NIHSS	Pure DL	88	79	ALROF(D)		
NMOSP	Profiled	0	1339	AL		
NMR	Pure DL	0	978	SRIQ		
NPO	Pure DL	0	16289	SHIN(D)	X	
NTDO	Pure DL	0	1133	SRIQ	X	
OAE	OWL Full	28	7001	SRIQ		
OBCS	Pure DL	33	1105	SROIQ(D)	X	
OBI	Pure DL	237	5846	SROIQ(D)		
OBI-BCGO	Pure DL	57	3547	SROIN(D)		

OBIWS	Pure DL	42	508	SROIQ(D)		X
OBOREL	Profiled	0	25	ALR+		
OCRE	OWL Full	45	967	ALCROIQ(D)		
OGDI	Pure DL	1095	754	SHIN(D)		
OGG	Pure DL	0	70026	SRIQ		
OGI	Pure DL	48	655	SRIQ(D)		
OGMD	Profiled	0	133	AL		
OGMS	Pure DL	20	385	SROIQ		
OGR	Profiled	0	38	AL		
OGSF	Pure DL	120	981	SROIQ(D)	X	
OMIT	Pure DL	0	3816	SRI		
OMRSE	Pure DL	21	206	ALCHOIQ	X	X
ONL-DP	Pure DL	0	1445	SHIQ		
ONL-MR-DA	Pure DL	84	1787	SHOIQ		
ONL-MSA	Pure DL	1826	3849	ALCHOIQ(D)	X	
ONLIRA	Pure DL	53	304	ALCHOQ(D)		
ONSTR	OWL Full	83	2554	SROIQ(D)		
ONTOAD	Profiled	10889	8364	ALH		
ONTODM-CORE	OWL Full	356	1925	SHOIQ(D)		
ONTODM-KDD	Pure DL	0	591	SHI		
ONTODT	Pure DL	278	489	SHOI		
ONTOKBCF	Pure DL	0	651	ALCHIF		X
ONTOMA	Profiled	27	373	ALC		
ONTOPNEUMO	Profiled	0	1153	ALH		
ONTOVIP	OWL Full	84	4718	SHOIQ(D)		
OOEVV	Pure DL	21	167	ALCO(D)		
OPB	Pure DL	0	924	ALCHIQ(D)		
OPE	Pure DL	0	85	ALCHIQ(D)		X
OPL	Pure DL	20	863	SHOIF		X
ORDO	Profiled	0	45825	ALE		
ORTHO	OWL Full	0	67	ALUHI(D)		
PAE	Profiled	0	2434	S		
PATHLEX	Profiled	0	1783	AL		
PATO	Profiled	0	2068	SH		
PCO	Pure DL	20	2467	SROIQ		X
PDO	Pure DL	0	366	ALUHI		
PDON	Profiled	0	1252	ALE		
PEAO	Pure DL	0	2802	ALCHQ(D)		
PECO	Profiled	0	557	AL		
PEDTERM	Profiled	0	1760	AL		
PHARE	Pure DL	0	403	ALCHIF(D)		
PHENX	Profiled	0	339	AL		

PHYLONT	Pure DL	2	210	ALCH(D)		
PIERO	OWL Full	288481	212	ALRI+		
PLIO	Pure DL	0	1055	ALE		
PMA	Profiled	0	10	AL(D)		
PMR	Pure DL	0	163	ALU		
PO	Profiled	0	2802	S		
PORO	Pure DL	2	963	SRIQ	X	
PPIO	Profiled	0	2892	ALE+		
PR	Pure DL	0	222268	S		
PROPREO	OWL Full	96	616	SHOIN(D)		
PROVO	Pure DL	2	135	ALCRIN(D)		
PSDS	Profiled	0	313	ALE+		
PSEUDO	Profiled	0	19	AL		
PSIMOD	Profiled	0	3587	ALE+		
PTO	Profiled	0	1490	ALE+		
PTRANS	Profiled	0	24	AL		
PW	Profiled	0	2001	ALE		
QIBO	OWL Full	683	1016	ALUIF(D)		
QUDT	OWL Full	116	369	SHIN(D)		
RB	Pure DL	0	423	ALEHIF+(D)		
REPO	Profiled	0	91	AL		
RETO	Pure DL	0	415494	SRI		
REX	Profiled	0	734	ALE		
REXO	Pure DL	0	463742	SRI		
RH-MESH	Profiled	0	432805	ALE		
RNAO	OWL Full	0	547	SRIQ		
RNPRIO	Profiled	0	76	AL		
RNRMU	Profiled	0	2243	AL		
ROO	Pure DL	29	151	ALCHI(D)		
RPO	Profiled	0	2047	ALF(D)		
RS	Profiled	0	5399	ALE		
RSA	Pure DL	4	22	ALUHOF(D)		
RXNO	Pure DL	0	1256	ALCH		X
SAO	Pure DL	0	2908	SHIN(D)		X
SBO	Profiled	0	641	AL		
SBOL	Profiled	0	47	ALE+		
SCHEMA	OWL Full	0	1996	ALU(D)		
SDO	OWL Full	110	2716	SHOIQ(D)		
SEDI	Pure DL	0	138	ALCHIQ(D)		
SEP	Profiled	0	193	AL		
SHR	Pure DL	0	417	ALH(D)		
SIO	Pure DL	0	2142	SRIQ(D)		

SITBAC	OWL Full	557	615	ALCON(D)		
SO	Pure DL	11	2445	SHI		
SOY	Profiled	0	1832	AL		
SPD	Profiled	0	843	ALE+		
SPO	Pure DL	0	641	ALERIF+		X
SPTO	Profiled	0	422	ALE		
SSE	OWL Full	2323	267	SHIF		
SSO	OWL Full	1474	210	ALIF(D)		
STATO	Pure DL	85	1654	SROIQ(D)	X	
SUICIDEO	Pure DL	110	346	SO		
SUICIDO	Pure DL	93	294	SHOI		
SWEET	OWL Full	3764	6210	SHOIN(D)		
SWO	OWL Full	13855	5142	ALRI+(D)		
SYMP	Profiled	0	840	AL		
SYN	Profiled	0	15353	ALF		
TAO	Pure DL	0	5186	ALERI+		
TAXRANK	Profiled	0	58	AL		
TEDDY	OWL Full	0	12562	SRIQ(D)		
TESTONTO	Profiled	1	3	AL		
TMA	Profiled	0	60	ALI(D)		
TMO	Pure DL	26	459	SRIN(D)		X
TOK	Pure DL	50	397	SRIQ(D)		
TOP-MENELAS	Profiled	0	1018	ALCH		
TRAK	Profiled	0	1829	ALER+		
TRON	Profiled	0	2065	ALEH		
TTO	Profiled	0	38639	AL		
TYPON	Pure DL	0	195	SHQ(D)		
UBERON	Pure DL	0	41218	SRIQ		
UNITSONT	Profiled	1	62	AL		
UO	Profiled	0	389	ALE		
VARIO	Profiled	0	402	ALE+		
VHOG	Profiled	0	1688	ALE+		
VIVO	OWL Full	13	1034	ALEHIN+(D)		
VO	Pure DL	68	12335	SROIQ		
VSAO	Profiled	0	453	ALER+		
VSO	Pure DL	0	763	SRIQ	X	
VT	Profiled	0	3929	AL+		
VTO	Profiled	0	106894	AL		
WB-BT	Pure DL	0	12372	ALC		X
WB-LS	Profiled	0	1282	ALEH+		
WB-PHENOTYPE	Profiled	0	2531	AL+		
WIKIPATHWAYS	Profiled	2	26	ALC(D)		

WSIO	Profiled	0	32	ALE
XAO	Profiled	0	5150	ALE+
XCO	Profiled	0	563	ALE+
XEO	Profiled	0	237	ALE
ZEA	Profiled	0	217	ALE
ZFA	Pure DL	0	11279	SHI

Table A.1: Full list of ontologies in the BioPortal snapshot we use as part of this thesis. PROF is the main profile category, ABox is the number of ABox axioms, TBox the number of TBox axioms, Expressivity is the corresponding Description Logic language, an X in the K1 category indicates that the ontology was used as part of experiment run K1 in Chapter 7; K2 analogously.

A.2 Method for RDFS detection

The original version of the method was written by Rafael Gonçalves [Gon14]. This is the updated version, including the check whether domain restrictions are restricted to class names.

Algorithm A.2.1: Method for RDFS detection

```
Data: Ontology O
Result: Boolean RDFS
RDFS = TRUE ;
AXIOMS = O.getLogicalAxioms() ;
while AXIOMS.hasNext() do
    AX = AXIOMS.next();
    if AX.type(SUBCLASSOF or SUBOBJECTPROPERTY) then
        if AX.getSub().isAnonymous() or AX.getSuper().isAnonymous() then
            RDFS = FALSE;
            break;
        end
    else if ASSERTION)) then
        do nothing;
    else if AX.type(PROPERTY-(DOMAIN or RANGE orASSERTION)) then
        if AX.getDomainOrRange().isAnonymous() then
            RDFS = false;
            break;
        end
    else if AX.type(CLASSASSERTION) then
        if AX.getClassExpression().isAnonymous() then
            RDFS = false;
            break;
        end
    else
        comment: some other axiomtype;
        RDFS=FALSE;
        break;
    end
end
```

A.3 OWL Reasoners

Table A.2 gives an overview of existing OWL reasoners. A more comprehensive analysis can be found in [MLH+15].

Table A.2: Overview of the OWL reasoners. SC stands for soundness/completeness, P for profile, O1 for OWL 1 and O2 for OWL 2. CALC is the main underlying calculus, EXP the highest expressive language supported. ACT indicates whether there is active development (B=Bugfixes, D=Active development, N=No development)

Name	Institution	SC	ACT	CALC	EXP
BaseVISor[MBK06]	VIStology, Inc.	P	B	Rete Network	NA
BUNDLE[RBLZ13]	Univ. of Ferrara	O2	D	Tableaux	SROIQ
CEL[BLS06]	Technische Universitt Dresden	P	N	Consequence-based	EL+
Chainsaw[TP12a]	Univ. of Manchester	O2	D	Modular Reasoner	SROIQ
Clipper[EOS+12]	Vienna Univ. of Technology	P	B	Query Rewriting	Horn-SHIQ
DBOWL[GM12]	Univ. of Malaga	O1	D	Relational Algebra and fixed-point iterations	SHOIN
DeLorean[BDGR12]	Not given	O2	D	Fuzzy	NA
DistEL[MHML15]	Wright State Univ.	P	D	Consequence-based	NA
DRAOn[DLZC13]	Univ. of Paris 8, IUT of Montreuil	O1	D	Compressed models	NA
DReW[XE11]	Vienna Univ. of Technology	P	B	Datalog Rewriting	EL++
ELepHant[Ser13]	Not given	I	D	Consequence-based	EL++
ELK[KKS14]	Univ. of Ulm, Germany	I	D	Consequence-based	EL+
ELOG[NNS11]	Not given	P	B	Integer Linear Programming	NA
FaCT++[TH06]	Univ. of Manchester	O2	D	Tableaux	SROIQ
fuzzyDL[BS08]	ISTI - CNR	I	D	Tableaux	SHIF
HermiT[GHM+14]	Univ. of Oxford	O2	N	Hypertableaux	SROIQ
jcel[Men12]	Technische Universitt Dresden	P	B	Consequence-based	EL+
JFact[Pal15]	Univ. of Manchester	I	D	Tableaux	SROIQ
Konclude[SLG14]	Univ. of Ulm, derivo GmbH	O2	D	Hybrid	SROIQV
LiFR[TDKM14]	Centre for Research and Technology Hellas (CERTH)	P	D	Hypertableaux	OWL DLP
Mastro[CGL+11]	Sapienza Univ. of Rome	P	D	Query Rewriting	DL-LiteA
MORe[RGH12]	Univ. of Oxford	O2	D	Modular Reasoner	SROIQ
ontop[KRRM+14]	Free Univ. of Bozen-Bolzano	P	D	Query Rewriting	OWL 2 QL
Pellet[SPG+07]	Clark & Parsia, LLC	O2	D	Tableaux	SROIQ
Racer[HHMW12]	Concordia Univ., Montreal, Canada; Univ. of Luebeck, Germany;	P	B	Tableaux	SRIQ
RDFox[MNP+14]	Univ. of Oxford	P	D	Datalog Rewriting	OWL 2 RL
RuQAR[BNJ14]	Poznan Univ. of Technology	P	D	Datalog Rewriting	
Snorocket[MJL13]	CSIRO	I	D	Consequence-based	EL++
TReasoner[GI13]	Tyumen State Univ.	O2	B	Tableaux	SROIQ
TRILL[ZBLR13]	Univ. of Ferrara	O1	D	Tableaux	SHOQ
TRILLP[ZBL+14]	Univ. of Ferrara	O1	D	Tableaux	ALC
TrOWL[TPR10]	Univ. of Aberdeen	P	D	Consequence-based	SROIQ
WSClassifier[SSD12]	Univ. of New Brunswick, Canada	I	B	Hybrid	ALCHOI

A.4 Supplementary materials Chapter 7

$\mathcal{T}$	$\mathcal{R}$	M1	DC	Res	M2	PP	CC	PR	ST	PO	Res	M3
CC	H	16.3	16.1	0.2	83.7	1.1	0.0	45.4	37.0	0.1	0.0	0.0
CC	J	16.1	15.5	0.5	83.9	7.6	0.0	0.0	75.9	0.3	0.0	0.0
CC	P	28.1	27.9	0.2	71.9	34.9	0.0	0.0	36.9	0.1	0.0	0.0
CCO	H	16.4	16.1	0.2	83.6	1.0	0.0	48.4	34.1	0.1	0.0	0.0
CCO	J	16.1	15.5	0.7	83.8	7.4	0.0	0.0	76.1	0.3	0.0	0.0
CCO	P	27.9	27.6	0.3	72.2	35.0	0.0	0.0	37.0	0.1	0.0	0.0
CD	H	22.0	8.9	13.1	78.0	2.5	0.5	47.8	26.4	0.8	0.0	0.0
CD	J	14.1	5.9	8.2	85.9	5.1	1.1	0.0	79.2	0.4	0.0	0.0
CD	P	14.7	3.4	11.2	85.3	21.9	2.2	0.0	60.1	1.1	0.0	0.0
MM	H	4.2	2.8	1.4	95.8	8.3	1.9	38.0	45.0	2.4	0.1	0.0
MM	J	3.7	2.5	1.2	96.3	12.0	2.8	0.0	79.9	1.5	0.1	0.0
MM	P	6.3	4.1	2.2	93.7	17.6	7.1	0.1	63.4	5.4	0.2	0.0
MON	H	0.7	0.0	0.7	98.5	4.2	0.4	63.2	25.3	5.3	0.0	0.8
MON	J	0.3	0.0	0.3	99.7	7.6	0.6	0.0	91.0	0.4	0.0	0.0
MON	P	0.4	0.0	0.4	98.0	9.6	2.3	0.0	74.5	11.6	0.0	1.6
MOR	H	3.8	3.7	0.1	96.2	4.6	0.3	49.9	35.8	1.1	4.5	0.0
MOR	J	2.1	2.0	0.1	97.8	5.9	0.0	0.0	88.7	0.3	3.0	0.0
MOR	P	2.8	2.8	0.0	97.1	44.4	0.1	0.0	49.2	0.6	2.9	0.0
MORA	H	3.7	3.6	0.1	96.3	5.1	0.1	51.0	38.4	1.8	0.0	0.0
MORA	J	2.4	2.2	0.1	97.6	6.2	0.0	0.0	90.8	0.6	0.0	0.0
MORA	P	3.3	3.3	0.0	96.6	22.1	0.1	0.0	72.9	1.5	0.0	0.0

Table A.3: Impact of sub-processes in % of overall classification time (OCT), experiment run K1. The columns in gray represent the three main modular classification stages: M1 pre-processing, M2 modular classification and M3 post-processing. DC is the decomposition time, columns PP, CC, PR, ST and PO stand for the sum of the contribution of the delegate reasoners: PP is the sum of pre-processing stages of all delegates (in % of OCT), CC the sum of all consistency checks, PR the pre-reasoning processing, ST the time spend traversing the class graph and PO the sum of all post-processing time measurements. The two columns labelled *Res.* represent the remaining time not accounted for my the preceding columns with respect to M1 and M2 respectively.

Figure A.1: Experiment K2, ontology name starting A to C. Breakdown of factors contributing to overall reasoning time, by primary delegate reasoner and ontology. X-axis is time spent, with the ratio between times being preserved across ontology. As we are only interested in comparing ratios, we omitted the axis labels. Dec is the decomposition time, PP (no DEC) is the remaining time needed for preprocessing (assigning ontologies to delegate reasoners, determining order), DEL-PP is the total time spent by delegate reasoners doing pre-processing, DEL-SCC is the total time spend consistency checking, DEL-PRP pre-reasoning processing and DEL-SST subsumption testing. Note that we have excluded the CD strategy on all plots because it was dominated by PP (no DEC) and rendered the other techniques unreadable (inefficient implementation).

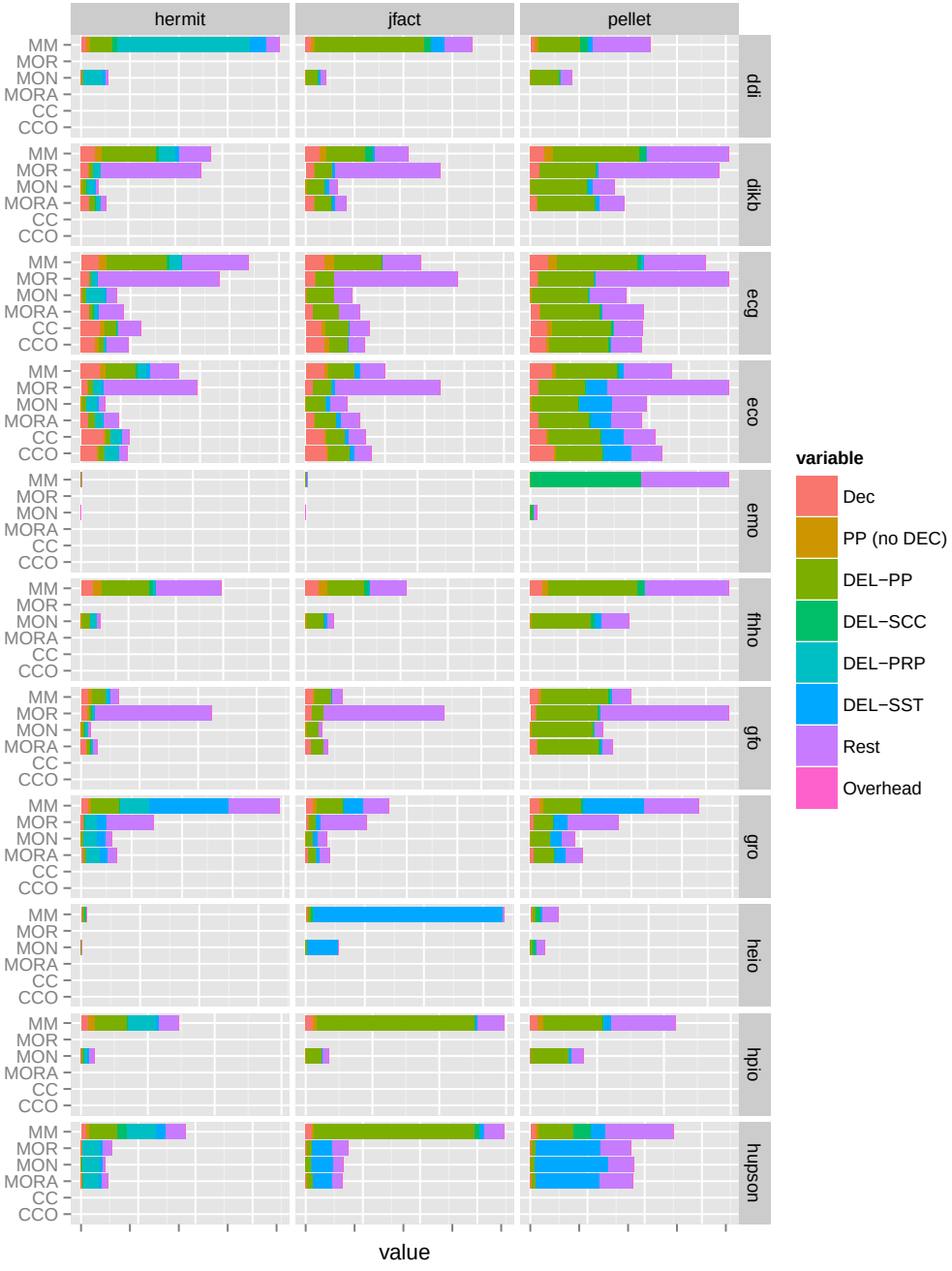


Figure A.2: Experiment K2, ontology name starting D to H. See Table A.1 for explanation of legend.

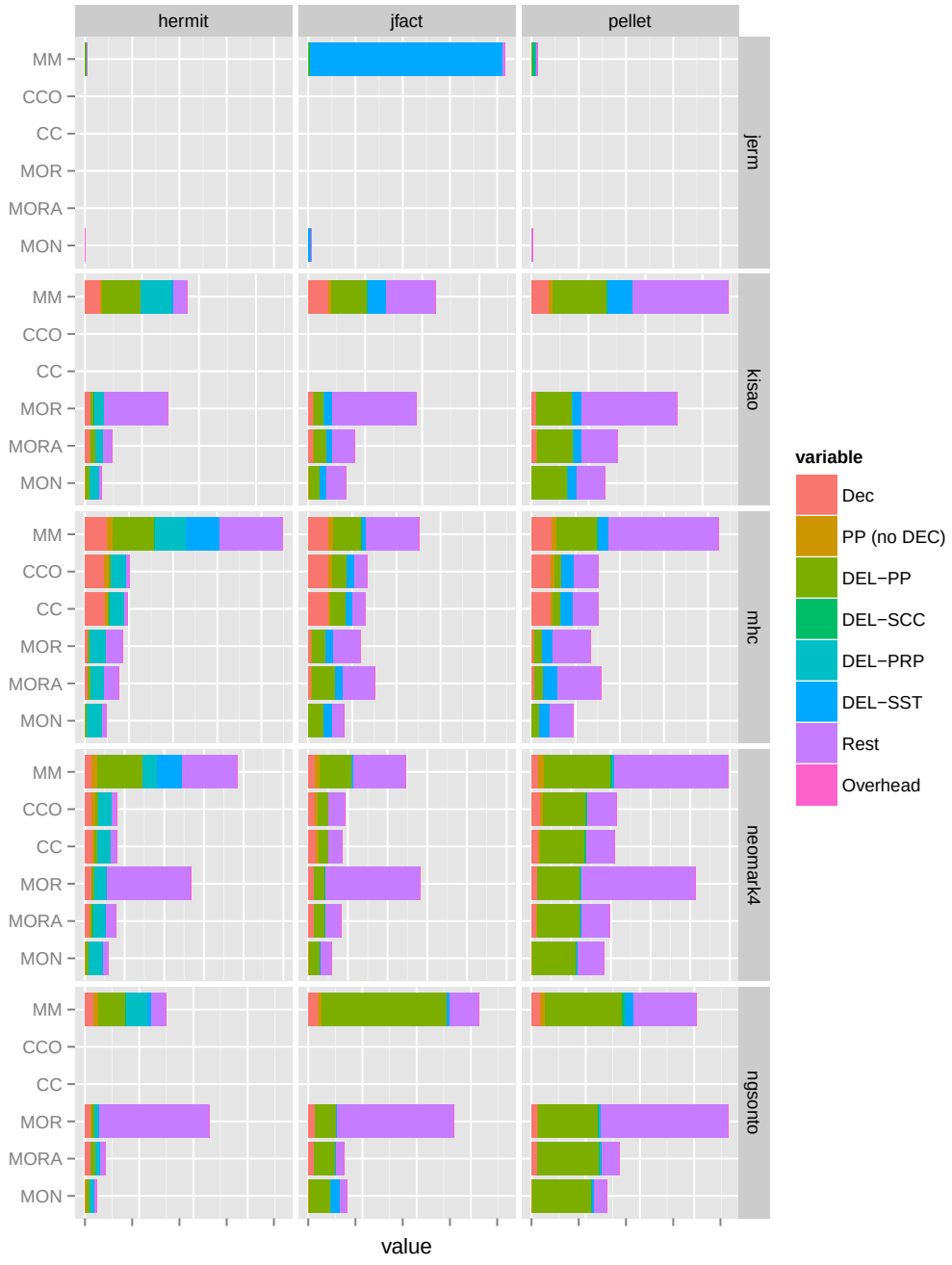


Figure A.3: Experiment K2, ontology name starting I to N. See Table A.1 for explanation of legend.

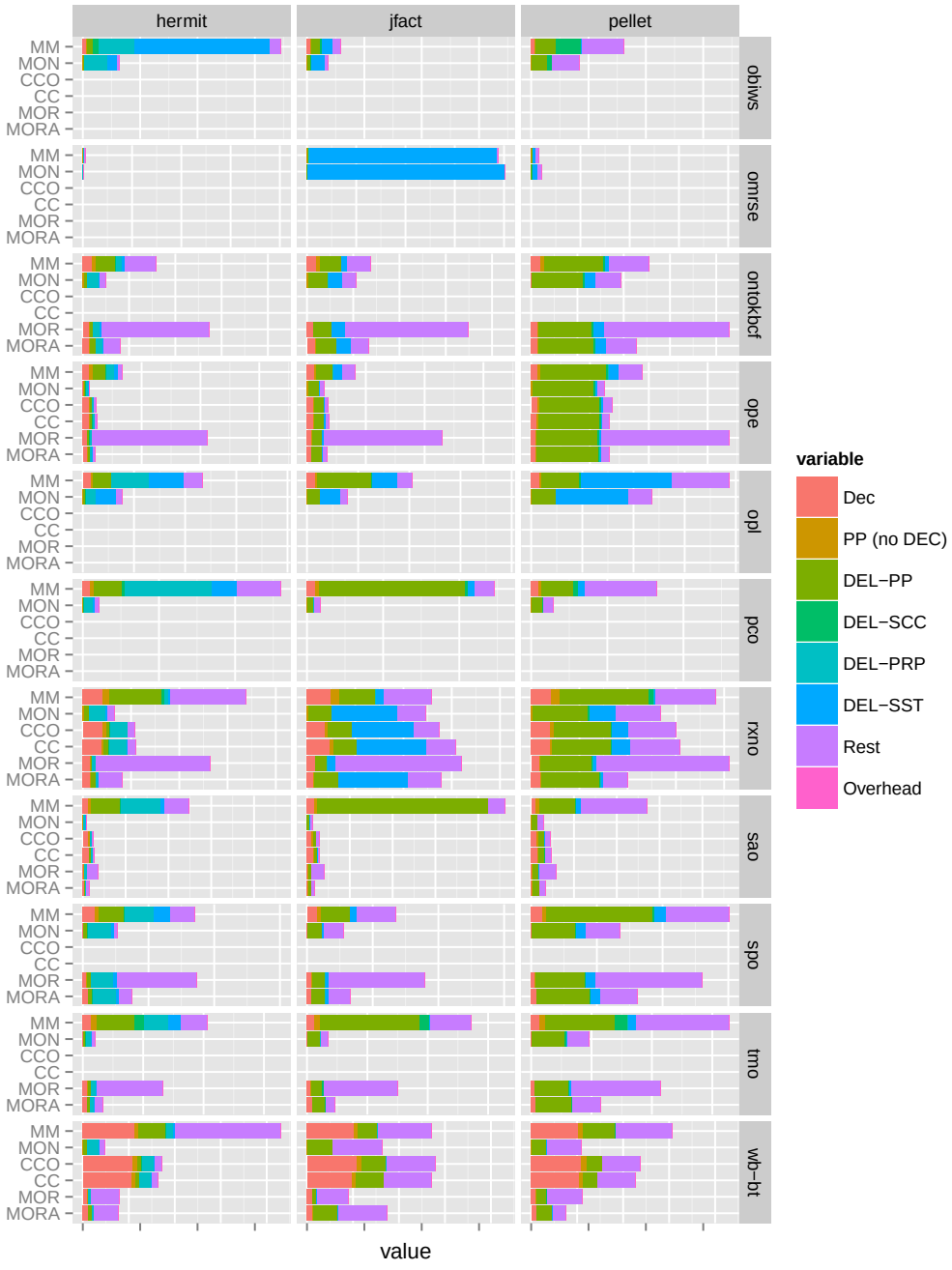


Figure A.4: Experiment K2, ontology name starting O to Z. See Table A.1 for explanation of legend.

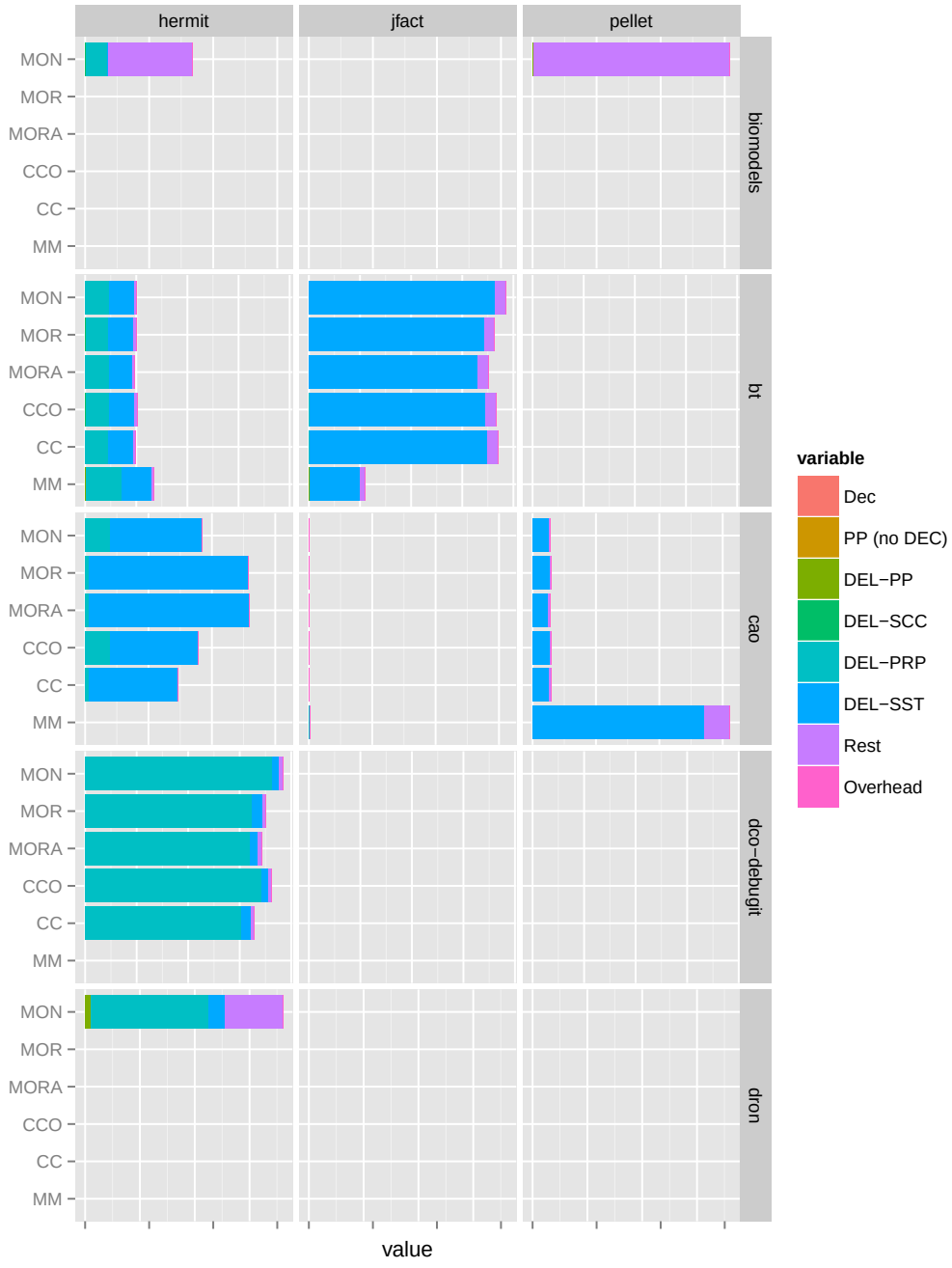


Figure A.5: Experiment K1, ontology name starting A to H. See Table A.1 for explanation of legend.

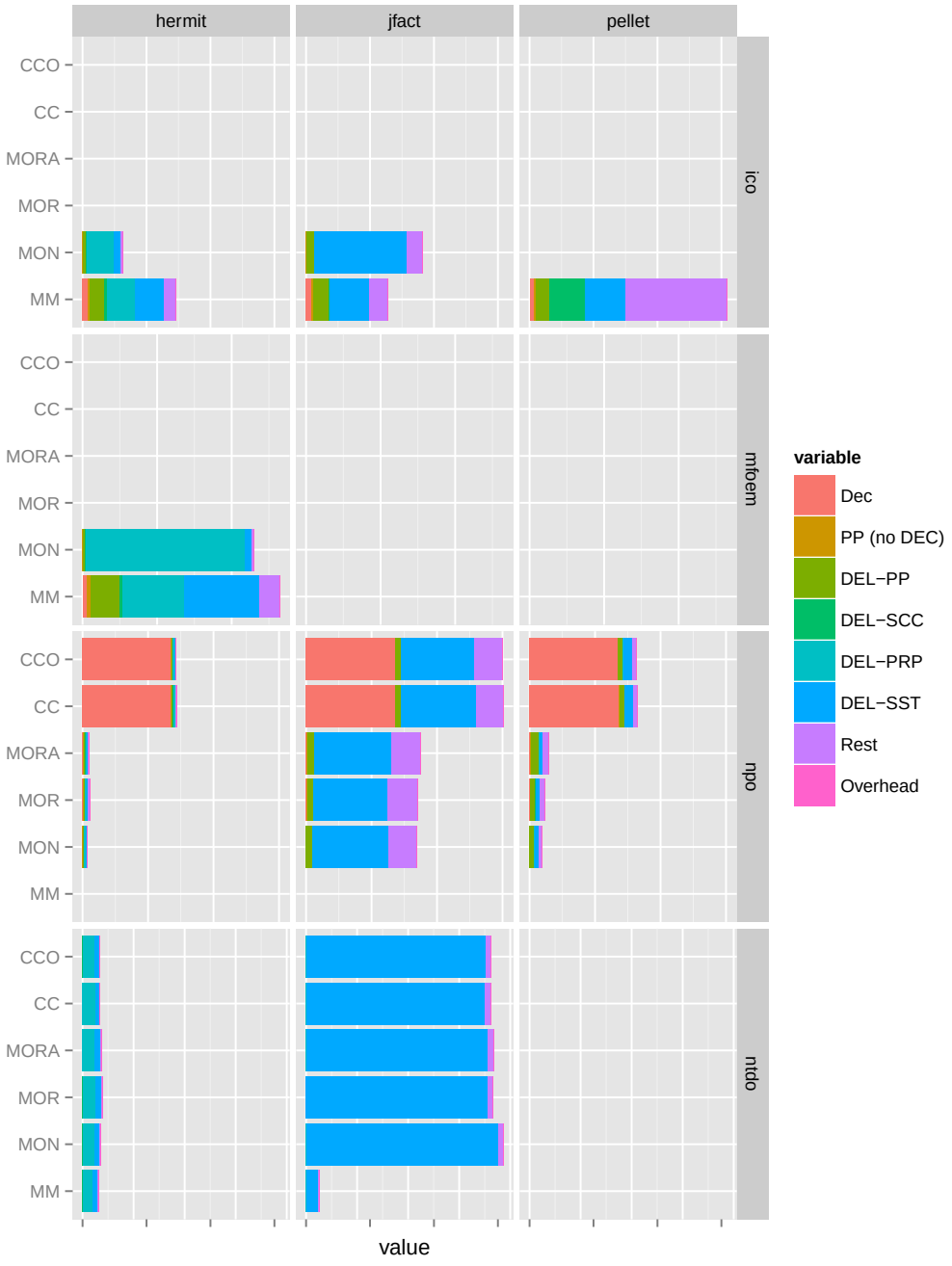


Figure A.6: Experiment K1, ontology name starting I to N. See Table A.1 for explanation of legend.

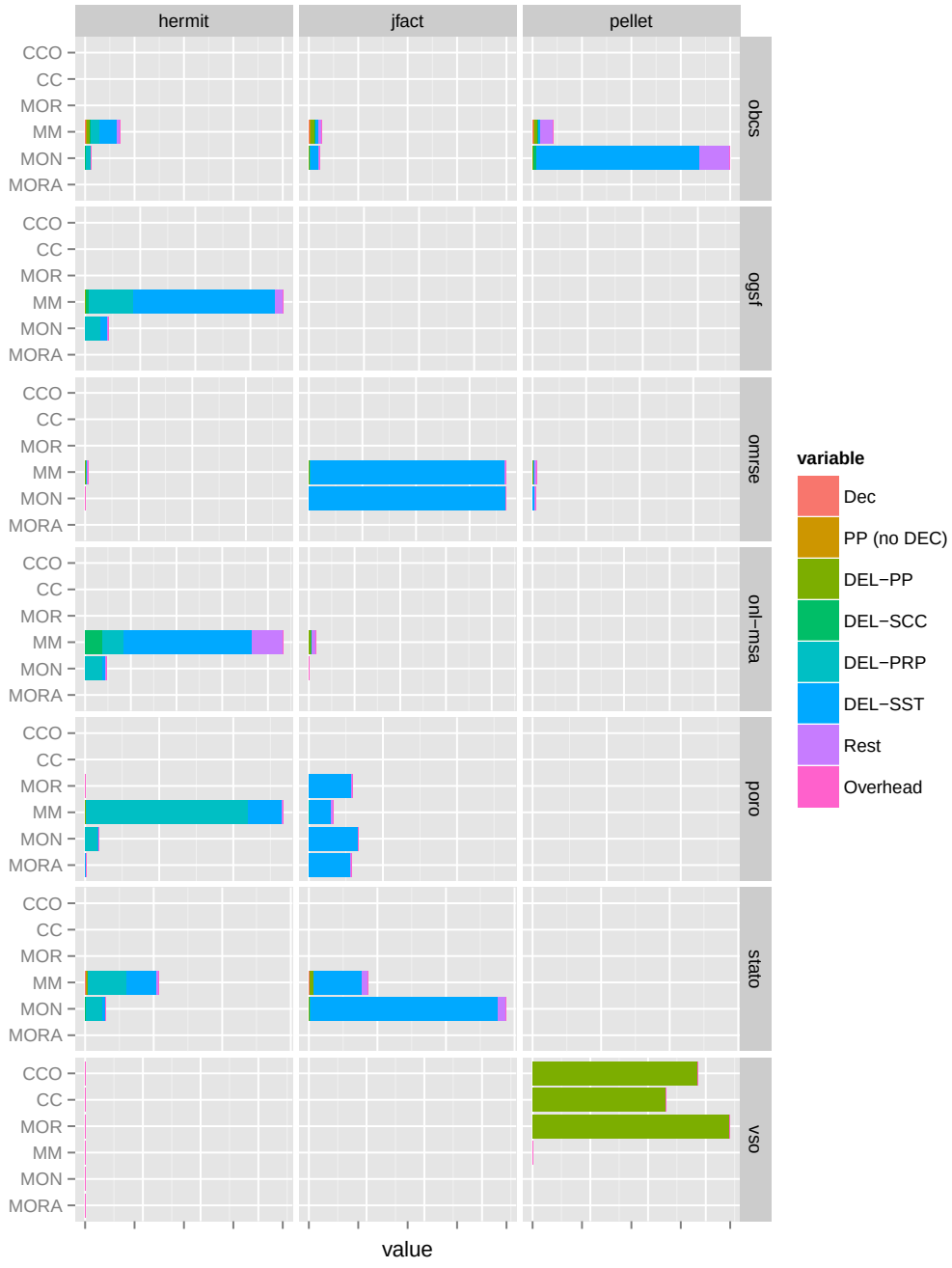


Figure A.7: Experiment K1, ontology name starting O to Z. See Table A.1 for explanation of legend.

Glossary

ABox The set of all class assertions and property assertions in the ontology (A for assertional).. 35

absorption Reasoner optimisation that attempts to reduce the high degree of non-determinism induced by general concept inclusions, for example by rewriting axioms.. 78

axiom A statement in OWL, for example $A \sqsubseteq B$, Functional(R), DisjointClasses(A, B).. 35

classification Reasoning task that determines, for every pair A,B of concept names in the ontology, whether A is a subclass of B.. 38

classification time Denoted $CT(\mathcal{O}, \mathcal{R})$. The time it takes to classify an ontology $\mathcal{O}$ by a reasoner $\mathcal{R}$ (informal).. 60

classification-preserving decomposition Sets of possibly intersecting subsets of the ontology that, if all are classified one-by-one, add up to the full classification of the whole ontology.. 51

conjunctive query answering Querying that goes beyond mere instance retrieval.. 38

consistency checking Reasoning task that determines the consistency of an ontology.. 38

decomposition-based reasoner A reasoner that performs classification on a classification-preserving decomposition of the ontology.. 60

entailment checking Reasoning task that checks, for a given axiom, whether it is entailed by the ontology.. 38

experiment run A single execution of a task such as classification on a specified machine.. 76

General Concept Inclusion A GCI or General Concept Inclusion is an axiom of the form $C \sqsubseteq D$, were C and D are (potentially complex) concepts. A TBox for example is a finite set of GCI's. 61

imports closure The set of ontologies directly or indirectly imported by a given root ontology.. 35

justification Denoted $\mathcal{J}^{\alpha}_{\mathcal{O}}$. The set of all justifications of an entailment α with respect to ontology $\mathcal{O}$, i.e. a minimal subset of $\mathcal{O}$ that entails α.. 60

logical axiom Denoted α. Subset of axioms that excludes annotation assertions and entity declarations.. 35

modular reasoner A reasoner that performs reasoning on a set of modules in the ontology.. 60

monolithic reasoner A reasoner that performs classification on the whole ontology rather than on its classification-preserving decomposition.. 60

nominal Given an individual a, we call {a} a nominal, denoting the a class whose only member is a.. 66

Pearson correlation coefficient Pearson's correlation coefficient is a measure of the strength of the association between the two variables.. 167

RDFS Basic schema language that allows for subsumption, assertions, domains and ranges.. 107

realisation Reasoning task that computes, for all concept names in the ontology, the named individuals that are members of it.. 38

signature Denoted $\widetilde{\mathcal{O}}$. The set of all names occurring in the ontology, including class names, property names and individuals.. 35

subsumption test A subsumption test between two concepts A and B as triggered as part of a traversal based classification procedure (such as Enhanced Traversal / Tableau) of $\mathcal{O}$. Typically denoted $ST(A, B, \mathcal{O})$.. 60

subsumption test measurement Denoted $ST(A, B, \mathcal{O}, \mathcal{R}, i)$. A subsumption test measurement between A ad B by reasoner $\mathcal{R}$ in $\mathcal{O}$, with i denoting the run.. 60

TBox The set of all logical axioms in the ontology excluding ABox axioms (T for terminological).. 35