DETECTION OF SYNTACTIC AND SEMANTIC REGULARITIES IN ONTOLOGIES

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

2013

By
Eleni Mikroyannidi
School of Computer Science

Contents

Word Count: 44293

List of Tables

List of Figures

Abstract

Ontologies are machine processable artifacts and the core structures of the Semantic Web. OWL (Web Ontology Language) is a W3C Recommendation language for developing ontologies; it is based on Description Logics, allowing for precise knowledge representation and sound and complete automated reasoning over the collection of axioms in an OWL document.

Although ontologies are useful for sharing terminologies, their design and reuse are difficult and time consuming processes. Despite the efforts of the community towards the development of OWL ontologies, there is a lack of methods and tools for reusing and inspecting ontologies, i.e., *reverse engineering methods*.

This thesis focuses on the area by investigating the **detection of regularities** in ontologies, for the purpose of abstracting sets of axioms into patterns that can be verified and reused. Its main contribution is the **Regularity Inspector for Ontologies (RIO)** framework, which implements methods to find *syntactic regularities* (repetitive structures in the *asserted axioms*) and *semantic regularities* (repetitive structures in the *entailments*) in an ontology. Regularity detection is achieved through the use of cluster analysis for detecting similarities in sets of axioms. This thesis provides experimental evidence for the effectiveness of regularity analysis for the inspection of patterns, and the discovery of modeling irregularities (often modelling errors) during quality assurance for real, large ontologies. In particular, empirical analysis showed that RIO could successfully detect regularities in ontologies, revealing the patterns adopted by the developers. It can be also used to trace pattern deviations as part of checking conformance to an intended design template during quality assurance of an ontology.

This work has been motivated by the existence of pattern based systematic development methodologies and the lack of methods for discovering patterns in existing ontologies—the natural complement of these pattern based development methodologies.

Declaration

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

Copyright

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor Dr. Robert Stevens for his constant support and guidance in fulfilling this PhD project. I would also like to thank my co-supervisor Prof. Alan Rector for his advice and support even after his retirement.

I would like to thank Dr. Luigi Iannone for helping me enormously in the planning of this research and for being equal to an advisor in this project.

I would like to thank Dr. Ignazio Palmisano for his critical contributions on the code optimisations and for his valuable suggestions on this work. Also I would like to thank him for his patience and for proofreading many times this thesis.

Special thanks to Dr. Dmitry Tsarkov for his valuable help on certain phases of this work and on his insightful comments on the thesis. Also I would also like to thank all friends and members from the Bio-Health Informatics Group and the Information Management Group in Manchester.

Finally, many thanks to all my friends who morally supported me throughout my studies. Last but not least, this PhD project is dedicated to my beloved family that always has been there for me in good and bad times.

Chapter 1

Introduction

Ontologies are the core models of the Semantic Web and a prominent component of the Linked Data paradigm [BHBL09]. OWL (Web Ontology Language) is a W3C standard language for developing logic based ontologies [W3C09]. OWL is built upon the foundations of *Description Logics* [BCM^{+}03]; a family of knowledge representation languages which are decidable fragments of the First Order Logic. There are many definitions of what an ontology is. From a Description Logic and computational point of view, an OWL ontology is a set of *axioms*, which are logical sentences which describe a logical theory about a field of interest. The power of OWL ontologies lies in their ability to generate inferences about the relationships among classes and instances in the ontology after automated reasoning [HS01, TH06a]. For this reason, ontologies are particularly well suited for research in areas with vast amounts of available data that need further exploration and exploitation, such as in biomedical research [RSN08, BS06].

A simple example of what an ontology looks like can be found in Figure 1.1. This example shows an intuitive graphical representation of axioms describing a small family[1]. The circles represent classes, like Man, Woman, Dog. Each class has a set of individuals like Quagmire, Lois, denoted with the diamond shape. Individuals are related to each other through properties like hasWife and hasPet. More details about basic notions of ontologies are described in Chapter 2.

On the downsides, the design of an ontology can be a difficult and time-consuming process. OWL is a highly expressive language and can be effectively used for the representation of complex domain knowledge such as biology, or medical terminology. However, it is quite difficult to form the axioms and describe

[1]'Inspired from the Family Guy series, `http://www.fox.com/familyguy/`

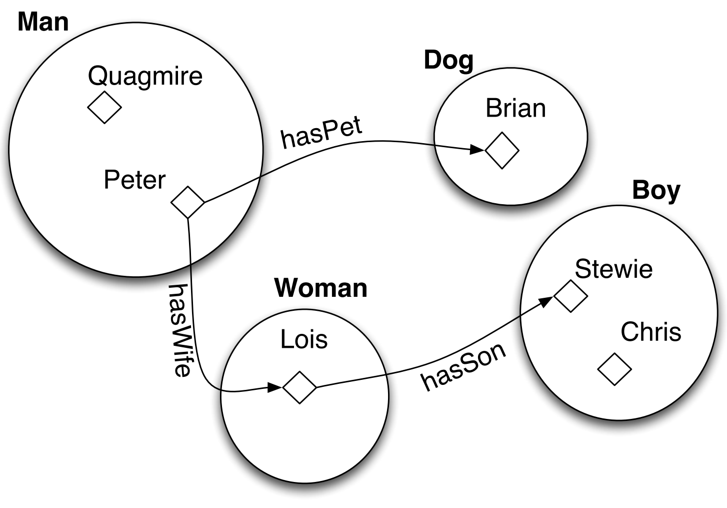


Figure 1.1: Representation of OWL Classes containing Individuals. The Individuals are connected with Properties.

concepts in the ontology without prior selection of a design techniques. This is similar to software development; the software engineer does not start implementing a program without thinking about a design for the program first; there is an initial plan on how objects and processes are connected to each other. This kind of decisions rely on the application for which the program or the ontology are going to be used; what is their purpose and motivation? In addition, after the development, the ontology needs to be properly managed and evolved so that it will be constantly aligned with changes that occur in the application domain in which it is used.

The work on effective design of ontologies [PSFGP10, RRB06, DCGPMPSF08] has led to the development of a large number of axiomatically rich ontologies. Editors like Protégé 4 and The NeOn Toolkit are professional environments for developing or editing an ontology by offering automated reasoning and explanation services for inferences, debugging of logical errors, multiple graphical views of the asserted and inferred ontology, modularisation services and so on. Methods for more effective ontology development, especially in the bioinformatics area, include libraries of design patterns and a meta-language for expressing them. The importance of design patterns in ontologies was analysed, among others, by Aranguren [Ara09]. Design patterns are modelling solutions for particular modelling problems. Ontology Design Patterns (ODPs) aim to facilitate the process of building axiomatically rich and rigorous ontologies. ODPs are similar to software design patterns; effective modelling solutions for common ontology modelling problems. The developers of ODPs are essentially trying to overcome design problems, to help ontology engineers select standard design decisions during the development of an ontology.

An example of an ontology fragment and related patterns, which will be explained in greater detail in subsequent chapters, is provided in Figure 1.2.

(1) Alanine *SubClassOf* hasCharge some Positive
(2) Aspartate *SubClassOf* hasCharge some Negative
(3) Cysteine *SubClassOf* hasPolarity some Non-Polar
(4) Glutamate *SubClassOf* hasPolarity some Polar

g1 = ?AminoAcid *SubClassOf* hasCharge **some** ?Charge
g2 = ?AminoAcid *SubClassOf* hasPolarity **some** ?Polarity

Figure 1.2: Four axioms from the Amino Acid ontology, describing four amino acids, and two patterns abstracting them.

All these efforts have led to the rapid development of ontologies, hosted in repositories like BioPortal[2] and TONES[3]. Such repositories, together with ontology searching services like Swoogle[4] and Watson[5], aim to facilitate ontology sharing; a key aspect of the semantic web. Thus, developers, instead of designing a new ontology from scratch, can reuse existing ontologies from repositories and extract fragments for the construction of a new ontology. This is similar to software reuse; code exists in repositories and developers do not have to implement a program from scratch, reusing fragments of existing code instead. This is a natural development as the corresponding field evolves and matures; in the past years

[2]`http://bioportal.bioontology.org/`
[3]`http://owl.cs.manchester.ac.uk/repository`
[4]`http://swoogle.umbc.edu/`
[5]`http://kmi-web05.open.ac.uk/WatsonWUI/`

there has been effort focusing on just writing programs for completing simple tasks, while today software engineering has evolved into agile development and higher reuse of software. This is where the need for reverse engineering methods starts to emerge: in software engineering a variety of techniques for inspecting code such as code slicing tools[6] [Kor02, RK02], abstraction techniques such as object orientation, functional programming, UML diagrams etc exist, to facilitate the process of understanding code.

In ontology engineering, the process of understanding an ontology, as part of reusing or maintaining it, is a difficult task. The user has to understand the meaning of each axiom as well as combinations of them to gain an idea of the underlying semantics. Such axioms are difficult to understand and sometimes non-intuitive, especially with regards to the conceptual implications of using an axiom or the consequences of applying automated reasoning after adding a new axiom to an ontology. In other words, the user has to achieve different levels of comprehension: understanding how the ontology was built, what design style has been used, and its underlying semantics and inferences. This task becomes even more difficult when dealing with ontologies that consists of hundreds of thousands of axioms. Even though there is a plethora of tools for ontology development there is a lack of methods and tools geared toward the inspection of ontologies and for quality assurance.

This research focuses on an aspect of ontology inspection, the detection of regularities. In this thesis, regularities are defined as repetitive structures in the asserted axioms and entailments of an ontology. The intuition behind it is to reveal the composition style of an ontology and highlight possible underlying patterns. The detection of such regularities can be helpful in many tasks such as:

1. **Inspection of the design style of an ontology:** Regularities can provide an abstraction layer on top of the low level axioms of an ontology. Since it is expected that particular design decisions were made during the development of an ontology, regularities can reveal such design by providing batch descriptions of axioms that reference similar entities in the ontology.

2. **Quality assurance:** The analysis of regularities of an ontology, and as a

[6]Program slicing is a technique for simplifying programs by focusing on selected aspects of semantics. The process of slicing deletes those parts of the program which can be determined to have no effect upon the semantics of interest. Slicing has applications in testing and debugging, re-engineering, program comprehension and software measurement.

consequence of its irregularities, can help the improvement of the ontology schema. These irregularities can be deliberate deviations of an ontology "coding" standard or discrepancies. These discrepancies cannot be traced by any reasoner as they are not logical errors. The detection and repair of such issues improves the quality of the ontology schema.

3. **Ontology integration:** Another scenario is a pattern-based approach for the integration of ontologies. This task is similar to the methods described in [ŠZSI10] and involves various alignment actions performed on the ontologies according to their underlying patterns. The detection of regularities can facilitate such procedure as it can highlight fragments of the ontology that have been designed in the same way.

More detailed usage scenarios are described in Chapter 3.3.

1.1 Research hypothesis and research questions

This research is on the broader area of the development of reverse ontology engineering methods. In particular, it focuses on the identification of repetitive axiomatic structures, named as *regularities* in an ontology.

The hypotheses of this research are:

1. Detection of syntactic regularities can reveal the compositional style of an ontology.

2. Detection of semantic regularities can reveal queries that can be asked about an ontology.

3. Analysis of regularities can help the inspection of irregularities in an ontology.

4. Regularities can provide an abstract view of the ontology.

The main hypothesis (1) of this thesis is that the recognition of syntactic regularities can be helpful for understanding the composition of an ontology in terms of such patterns, as it can reveal parts of the ontology that were designed in similar ways. In the field of ontology engineering, the use and recognition of patterns is important when authoring an ontology, in order to understand it and assure that it conforms to guidelines and intended patterns.

The second hypothesis (2) can be considered as an extension of the first hypothesis; the detection of semantic regularities can provide a summary of the information that can be inferred about similar entities in the ontology. In addition, the detected regularities in the entailments of the ontology can reveal queries that can be asked about the ontology.

The third hypothesis (3) is related to the analysis of the detected regularities. The claim made here is that the analysis of regularities can help a more systematic inspection of irregularities in an ontology. These irregularities can be either deliberate design deviations or design defects in an ontology. An intentional design deviation can be motivated by the need for a particular feature, or to work around a performance issue, where strict adherence to the pattern would have caused great cost when reasoning, and this choice should be documented for future developers, so that the intent is preserved. On the other hand, an unintentional deviation might be the consequence of a misunderstanding, and as such, basically a modelling error.

Finally the fourth hypothesis is related with the inspection of regularities in an ontology and the abstraction they create in the axioms and entailments of an ontology. Regularities can summarise repetitive structures in an ontology represented through generalisations; axioms with at least one meta-linguistic variable.

The validation of these hypotheses initially involves the evaluation of the detected regularities; whether they are valid and meaningful. That covers the assessment, through metrics, of the quality and validity of the detected regularities, and comparison with different independent structures of an ontology.

Each hypothesis is related with a number of research questions, which are answered in the remaining chapters of this thesis. These are:

- Questions related to hypothesis 1
  - How can syntactic regularities be detected in ontologies?
  - How can the validity of the detected syntactic regularities be assessed?
  - How can regularities reveal the compositional style of an ontology?
- Questions related to hypothesis 2
  - How can semantic regularities be detected in ontologies?
  - How can the validity of the detected semantic regularities be assessed?

  - How can semantic regularities reveal queries about an ontology?

- Question related to hypothesis 3
  - How can the analysis and inspection of regularities reveal irregularities in an ontology?

- Question related to hypothesis 4
  - How can regularities provide a summary of the ontology?

1.2 Contribution of this Thesis

The broad aim of this thesis is to advance the methods of authoring ontologies and in particular methods of revealing the compositional style of an ontology. To achieve this, this research focuses on the detection of regularities in ontologies as part of an inspection process to help authoring and makes the following contributions:

- **Explanation of the concept of regularities, irregularities, patterns and design patterns.** Even though the term *pattern* has been used in the field before, there are different interpretations of it. This thesis gives a description of regularities, and clarifies their difference from patterns and design patterns.

- **Computation of syntactic regularities.** One of the main contributions of this thesis is the computation of syntactic regularities, which are repetitive structures in the asserted axioms of an ontology.

- **Computation of semantic regularities.** Another important contribution of this thesis is the computation of semantic regularities, which are repetitive structures of a selected set of entailments in the ontology.

- **Implemented tools for the inspection of regularities.** A prototype tool has been developed for the computation as well as the inspection of regularities. In particular, a plugin with multiple views of clusters of similar entities and their description with generalisations along with stats for the generalisations has been developed for a professional ontology suite, Protégé 4[7].

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

- **Evaluation of the framework.** The framework for the detection of syntactic and semantic regularities in the ontology has been evaluated using a variety of strategies. These include validation through two different types of criteria:
  - **Internal criteria.** As the computation of regularities is based on clustering, there are metrics for assessing the quality of the results such as the compactness and separation of the clustered data.
  - **External criteria.** This approach tests whether the detected structures are meaningful by comparing the results with an independent partition of the data. This can be further divided into the following approaches:
    - Comparison with patterns described in the documentation of the ontology.
    - Projection and validation of the syntactic and semantic regularities against the modular structure.
- **Inspection of irregularities.** This thesis provides experimental evidence of the inspection of irregularities in an ontology as part of the analysis of the detected regularities. These were classified as either deliberate deviations from an initial pattern or as design defects.

1.3 Outline of Thesis

Chapter 2 presents background knowledge for the research area. It includes a discussion on the area of ontology engineering and ontology comprehension. Also, it describes different aspects of ontology comprehension using the software comprehension analogy and shows how different methods and tools in ontology engineering can be used to facilitate comprehension of an ontology. Finally, it presents background work on patterns and introduces the notion of regularities.

Chapter 3 presents the architecture of a framework for detecting regularities in ontologies, RIO, the main contribution of this research. It also describes some usage scenarios in which this framework can be potentially useful.

Chapter 4 describes the computation of syntactic regularities in ontologies, which is the first contribution of RIO. It presents the algorithms for the computation of the regularities and describes modelling decisions that were made for their computations.

Chapter 5 describes the computation of semantic regularities in ontologies, which are repetitive structures in a selected set of entailments in an ontology. It describes the details on the extraction of an entailment set from an ontology and details of the algorithms for computing semantic regularities. This is the second contribution of RIO.

Chapter 6 describes the evaluation methods for assessing the validity and quality of the detected regularities. This involves the inclusion of different types of criteria such as metrics and validation through other independent structures in the ontology.

Chapters 7 and 8 present a number of experiments and evaluation results. The experiments demonstrate the detection of syntactic and semantic regularities in ontologies as well as the inspection of irregularities in a variety of ontologies. The layout of these Chapters is symmetric; Chapter 7 presents the experiments on the syntactic regularities, while Chapter 8 presents the corresponding experiments for the semantic regularities on the same selection of ontologies allowing the comparison with the results of syntactic regularities.

Chapter 9 discusses outcomes of this research and reviews this thesis' contributions with respect to the evaluation results. It ends with pointers for future research.

1.4 Published Work

The work of this thesis is supported by the following journal, conference and workshop publications:

- [MISR11] E. Mikroyannidi, L. Iannone, R. Stevens, and A. Rector. Inspecting regularities in ontology design using clustering. *The Semantic Web, ISWC 2011*, pages 438-453. Springer, 2011.

- [MISR12] E. Mikroyannidi, L. Iannone, R. Stevens, and A. Rector. Inspecting regularities and irregularities in SNOMED-CT. *In Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences, SWAT4LS 2011*, pages 76-83, New York, NY, USA, 2012. ACM.

- [MIS12] E. Mikroyannidi, L. Iannone, and R. Stevens. RIO: The Regularities Inspector for Ontologies Plugin for Protégé 4. In *3rd International Conference on Biomedical Ontology (ICBO)*, June 2012.

- [MMIS12] E. Mikroyannidi, N. A. A. Manaf, L. Iannone, and R. Stevens. Analysing syntactic regularities in ontologies. *In 9th OWL: Experiences and Directions Workshop (OWLED)*, 2012.

- [MSIR12] E. Mikroyannidi, R. Stevens, L. Iannone and A. Rector. Analysing Syntactic Regularities and Irregularities in SNOMED-CT. In *the Journal of Biomedical Semantics.*

- [MSR11] E. Mikroyannidi, R. Stevens, and A. Rector. Identifying ontology design styles with metrics. *In 7th International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Bonn, Germany, 2011.

Chapter 2

Ontology Engineering and Reverse Methods for Comprehension

This chapter introduces key ideas and definitions as well as the general scope in which this research is placed, including the presentation of fundamental notions and nomenclature that will be used in the remaining chapters. Section 2.1 is a short introduction to ontology engineering and the ontology comprehension problem. It can be used as an outline of this Chapter and as a map for introducing the key ideas that are discussed and explained later on.

2.1 The Landscape of Ontology Engineering

Ontologies are machine processable artifacts, useful for expressing and sharing domain knowledge [BCM$^+$03]. As mentioned in Chapter 1, OWL is a W3C standard language for developing logic-based ontologies. The family of OWL (1 and 2) languages is based on Description Logics, allowing for inferences to be made based on a set of asserted facts, named as *axioms*; two of the OWL languages, OWL DL and OWL 2 DL, have this notion expressed in their name, where DL stands for Description Logics. In OWL, axioms are built using OWL constructs, combined through operands. In a nutshell, they are logical sentences that describe a knowledge domain. Details about OWL constructs are described in Section 2.3.

Axioms can be written using a variety of OWL syntaxes. To give a simple example of an ontology expressed in OWL using the DL Syntax, consider the following ontology $\mathcal{O}$:

$\mathsf{Car} \sqsubseteq \mathsf{Vehicle}$ (1)
$\mathsf{Sports\_car} \sqsubseteq \mathsf{Car}$ (2)

These two axioms state that a Car is-a a Vehicle and a Sports_car is-a Car. The logical inference that follows from these two axioms is:

$$\mathcal{O} \models \mathsf{Sports\_car} \sqsubseteq \mathsf{Vehicle} \quad (a)$$

Expression (a) is also called an *entailment*. This is only one example of an entailment from this example ontology; in theory, infinite inferences can be made [BCM+03]. In practice, a software program, called a *reasoner*, is used for working with the ontology; it initially checks for logical errors and then generates inferences like (a), starting from the *asserted* facts, like axioms (1) and (2).

The OWL languages, up to OWL 2 DL, are characterised by the availability of efficient reasoning algorithms - meaning that the reasoning tasks are decidable, and in the average case reasoning can be performed quickly. More details on how reasoning is performed are described in Section 2.4.

Thus, ontologies can be efficient solutions for describing detailed domains and sharing terminologies, e.g.,in biology [BR04], health sciences [BM06] etc, as the developers do not have to assert all the required information manually, but large amounts can be left implicit and inferred from the asserted axioms by a reasoner. In addition, OWL provides a variety of constructs and features for describing a domain, allowing for the inclusion of more complex axioms.

The establishment of OWL as a W3C standard has contributed to the rapid development of ontologies for many disciplines and especially bioinformatics [BS06]. As an example, Figure 2.1 shows the use of word 'ontology' in PubMed[1] publications since 2000. The number of articles related with ontologies has rapidly increased over the last years and many leading journals in Bioinformatics and Computers in Biology and Medicine area, have devoted special issues on ontologies and regularly present papers on ontology applications.

[1]`http://www.ncbi.nlm.nih.gov/pubmed`

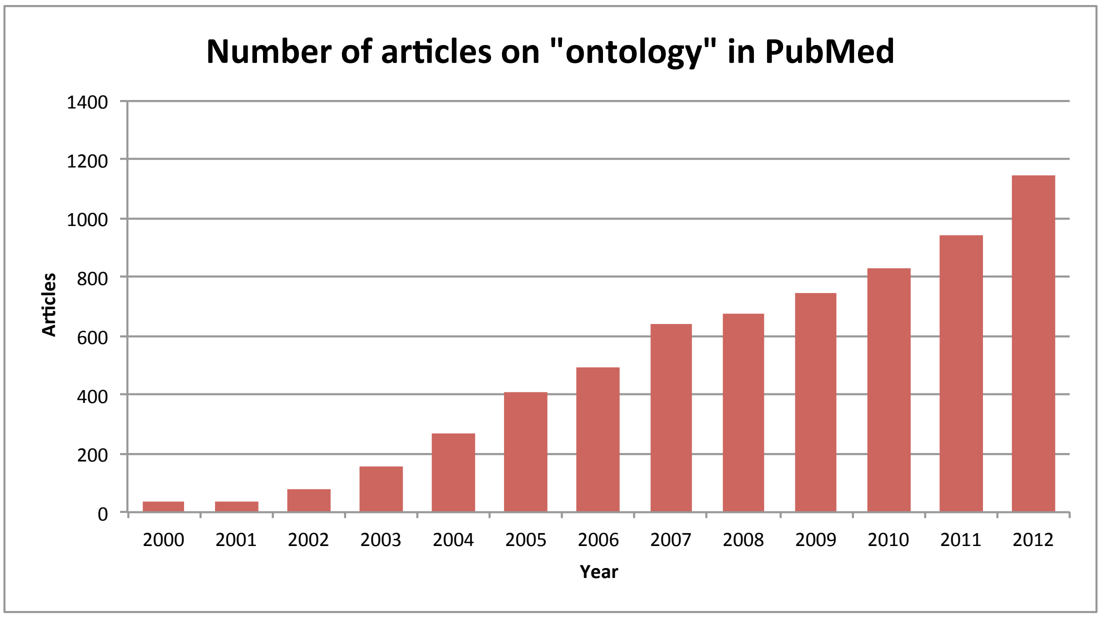


Figure 2.1: Growth of ontology papers in PubMed since 2000.

2.1.1 Problems with OWL and Ontology Engineering Solutions

From early years, the development and reuse of ontologies led to the development of ontology engineering methods [NH97, UHW+98]. However, with the development of ontologies and of the ontology engineering field, the necessity of better engineering solutions emerged [PSFGP10, RRB06, DCGPMPSF08, EF08, FLC10].

Ontologies describing complex domains, such as biology, anatomy and diseases, have being developed for the last 10 years [BS06] in the context of large projects. A few examples are the Gene Ontology (GO)[2] [Con04], SNOMED-CT[3], NCI thesaurus[4] and FMA[5] [RMJ+03]. Many collaborative efforts amongst different communities flourished the development of such terminologies. Such ontologies consist of hundreds to thousands of axioms of varying complexity. NCI thesaurus is one of the largest ontologies that covers numerous cancer research related domains [SdCH+07]. One of its latest versions released on 09/2011[6], consists

[2]http://www.geneontology.org/
[3]http://www.connectingforhealth.nhs.uk/systemsandservices/data/uktc/snomed
[4]http://ncit.nci.nih.gov/
[5]http://sig.biostr.washington.edu/projects/fm
[6]http://www.cs.man.ac.uk/~goncalvj/ncit/asserted.html

of 1 ,262 ,066 axioms. Since much effort has been spent on these ontologies, sharing and reusing such resources is easier than building new ones from scratch. For this purpose, ontology repositories like NCBO BioPortal[7] were developed to allow the sharing of biomedical and bioinformatics terminology. However, the reuse and extension of an ontology is not an easy task, especially because, as ontologies grow in size and complexity, they become more difficult to manage and maintain. SNOMED-CT is such an example; many efforts have focused on the maintenance and quality assurance of the ontology [RBS11, CSKD04, WHM$^+$07, RBK08]. More details on quality assurance are presented in Section 2.7.1.

Even though OWL is a very expressive logical language, as we will see in Section 2.2, it has been observed that expressing knowledge in OWL is a difficult task [RDH$^+$04], even for simple domains such as Pizza, the subject of a famous ontology used in many tutorials [RDH$^+$04].

Ontology editors and other tools have been developed to facilitate ontology development as we will see in Section 2.6, but most of these still offer little more than basic functionality for writing axioms in an ontology and basic visualisations of the structure of an ontology.

In addition, the unpredictable "behaviour" of an ontology with regard to its entailments has proved to be an issue for ontology development; this is because the misuse of OWL constructs and their combination resulted in unpredictable entailments and in logical errors that could not be resolved with manual inspection of the axioms [RDH$^+$04]; manual inspection is limited, in this context, both by the scale of ontologies composed of hundreds of thousands of axioms, and by the difficulty, for a human, to understand at once all the consequences of adding an axiom to an ontology.

Even worse, some of these errors will make the ontology inconsistent, i.e., no model could exist for this ontology. Details about inconsistency are described in Section 2.4. While detecting an inconsistency is a standard task for reasoners, and so it does not require human intervention for detection, repairing an inconsistency is much more difficult, since there can be more than one cause, and identifying the root issue is a complex task [PSK05].

This consideration leads to another key aspect of ontology engineering: *ontology debugging.* In particular, the problem of fixing unsatisfiable concepts and inconsistent ontologies often cannot be resolved with manual inspection of an

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

ontology. Theories and tools for dealing with ontology debugging have made this task easier [PSK05]. In addition, methods for providing explanations to various entailments and helping users to trace logical errors in ontologies have been developed [KPHS07, HP10, HPS10a, HPS10b]. Today, these ontology debugging and explanatory services have become an important feature of advanced ontology editors like Protégé 4[8] and The NeOn Toolkit[9]. All these are discussed in Section 2.6.2.

Module extraction and integration with other ontologies is a more advanced task concerning ontology reuse; it requires the understanding of an ontology for selecting the corresponding signature. The modularisation of an ontology would ensure the extraction of a desired part of an ontology for reuse in a different ontology. We discuss different types of modularisation algorithms in Section 2.5.

In addition to module extraction, collaborative efforts on the development of an ontology brought to attention another problem of ontology engineering; the development of methods for tracking ontology versions and avoiding conflicts when different people are editing an ontology [SEA$^+$02, HJ02].

All the above tasks, as most tasks involving ontology use and authoring, have a common characteristic, which is the need to understand an ontology, i.e., *ontology comprehension*. Ontology comprehension is an important and difficult task [NCA08, GWS07]. The process of understanding ontologies can be compared to the process of understanding code, its structure, organisation and purpose, in software engineering. The OWL language can be compared with similar low level programming languages such as assembler, as it represents the lowest level for ontology semantics; the components below OWL, such as its serialization and storage solutions, do not have semantic value, i.e., the same ontology can be stored as a file or in a database, or in any of a set of syntaxes, and each ontology has exactly the same interpretation. To the best of our knowledge, so far there is no framework combining ontology engineering methods for the purpose of ontology comprehension.

Different renderings of OWL ontologies, such as the OWL Functional Syntax [MPSP$^+$09] and the OWL Manchester Syntax [BVHH$^+$04], which are described in more detail in Section 2.3, can provide a more readable rendering of

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

[9]`http://neon-toolkit.org/`

an ontology but it is not enough as a standalone service to achieve comprehension. In addition, efforts on the interpretation of axioms in the ontology into natural language as an attempt for a more understandable format have been reported in [SMW$^+$11, RD00, GA07, Sch09]. All the above ontology engineering solutions whose initial purpose was not ontology comprehension as well as new ones could be integrated to *support ontology comprehension.* As it is discussed in Section 2.6.1, similar software engineering approaches were taken in the software engineering field for tackling the corresponding problem of software comprehension; the understanding of code. All these methods that can be used in the scope of comprehension are named as *reverse engineering methods.* More details about software reverse engineering and ontology reverse engineering are described in Section 2.6.

The reuse of ontologies can simplify the *comprehension* of an ontology, since a commonly used ontology will need to be understood once by a developer, and all subsequent reuses will require less study. This task involves initially the inspection of an ontology, identification of *patterns* and understanding of the axioms and their implications. Pattern injection and pattern recognition in ontologies are important aspects of ontology comprehension and the main focus of this thesis. The usage of patterns as ontology engineering solutions are thoroughly described in the following section as well as in Section 2.7.

The Role of Patterns in Ontology Engineering

Pattern recognition is one of the most critical skills in intelligent decision making, as it can help in predicting the evolution of a system with a high degree of accuracy [TK06]. In software engineering, pattern recognition has been used in software comprehension frameworks for the creation of code abstractions, facilitating inspection processes [Bro81].

In ontology engineering, patterns have so far been used in a different context. Ontologies as self-standing entities are Knowledge Representation artifacts: axioms express patterns of knowledge in a particular domain [BCM$^+$03]. Efforts towards knowledge extraction from text and its representation with ontologies are reported in [PRG$^+$09, JKS$^+$11]. Existing work on ontology patterns refers mainly to the development of ontologies based on patterns. Moreover, design patterns, meaning best design practices, are used often in common knowledge representation problems [GP09].

However, patterns can exist without being necessarily the result of the application of a design decision to a modelling problem; they can emerge as developers reapply the same process unconsciously, reinventing the same solution at different points in the development. Finding and formalising such patterns can be done through the RIO framework (introduced in Chapter 1 as the main contribution of this thesis).

Patterns can be considered the design templates that developers use when describing a set of entities in an ontology. These patterns can be expressed in various ways such as through spreadsheets, general documentation, conceptual schemas, etc [PRG+09, JKS+11]. Their instantiation in the ontology is expected to give rise to repetitive structures. These repetitive structures are axioms and sets of axioms with similar design. There are methods and tools for ontology development that allow to design ontologies using patterns; however, the detection of patterns in ontologies is still a novel area, and this thesis provides tools to advance the exploration in this research subject. Some prior work towards this direction is reported in [JŁŁ05] where the authors detect patterns using DL rules. However the detection of the patterns is not unsupervised.

This thesis deals with the problem of pattern recognition in ontologies. The contribution can be framed in the field of reverse ontology engineering methods: the methods described in this thesis can be used in a framework as a supporting tool for the comprehension of an ontology, of which RIO is a first draft.

Summary

This section described briefly aspects of ontology engineering. It also introduced issues with ontology use and reuse as a result of a lack of systematic reverse ontology engineering methods and tools. Sections 2.2, 2.3 and 2.5 describe in more detail fundamental notions of ontology engineering while Section 2.6 describes in more detail the problem of ontology comprehension. Section 2.6.2 describes aspects of the problem and, starting from software comprehension frameworks, draws an outline of an ontology comprehension framework. Finally, Section 3.1 gives the main definitions around different notions of patterns.

2.2 Description Logics

This section presents the basic notions and nomenclature for Description Logics and OWL.

Description Logics (DLs) are a family of logic based knowledge representation languages [BCM+03]. They are *decidable* fragments of First Order Logic (FOL). DL is used in artificial intelligence for formal reasoning on the terminological knowledge of an application domain.

This logical foundation provides for solid reasoning algorithms, which allow OWL ontologies to be used to draw sound and complete inferences about the knowledge domain they describe. The most notable application outside Computer Science is in bioinformatics and medical informatics, where DL assists in the codification of medical and biological knowledge.

The basic building blocks of DL are:

- **Concepts:** A concept is a unary predicate in FOL and is defined as a *class* in OWL. A class in OWL is defined as a group of individuals.
- **Roles:** They are named as *properties* in OWL and they are binary predicates in FOL.
- **Individuals:** An individual in OWL is defined as a single object. In FOL individuals represent constants.

Basic Constructs and Complex Concepts

Table 2.1 shows some basic DL constructs. C and D in Table 2.1 are concepts while R is a role. Complex concepts can be created as a combination of particular constructs and concepts. For example, the complex concept $\mathsf{Woman} \sqcap \mathsf{Artist}$ is a conjunction describing individuals who are instances of Woman and Artist; in an ontology, for example, there might be an individual identified with the name F representing Frida Kahlo[10] and belonging to this class expression; this is the OWL name for a composite class obtained through composition of DL constructs.

[10] `http://en.wikipedia.org/wiki/Frida_Kahlo`

Constructor	**DL Syntax**	**Example**
Top concept	$\top$	
Bottom concept	$\bot$	
Concept negation	$\neg$	$\neg$Child
Concept intersection	$C \sqcap D$	Woman $\sqcap$ Artist
Concept union	$C \sqcup D$	Cat $\sqcup$ Dog
Existential restriction	$\exists R.C$	$\exists$hasPet.Dog
Universal restriction	$\forall R.C$	$\forall$hasChild.Boy
TBox axioms		
Concept inclusion	$C \sqsubseteq D$	Car $\sqsubseteq$ Vehicle
Concept equivalence	$C \equiv D$	FamilyGuy $\equiv$ Man $\sqcap$ $\exists$ hasFamily.Family
ABox axioms		
Concept assertion	$a : C$	Brian : Dog
Role assertion	$(a, b) : R$	(Peter, Brian) : hasPet Peter has pet Brian

Table 2.1: DL constructors and examples.

Axioms

Axioms are logical statements about a knowledge domain. Axioms built with concepts, roles, individuals and constructors. Thus, axioms are logical statements relating concepts, roles and individuals with a (sub)set of the constructors mentioned above. There are three categories of axioms:

- **TBox axioms** are Terminological Box statements defining relationships between concepts, concepts and roles, and between roles and datatypes. TBox axioms that define concept inclusions are called subclass axioms in OWL. An example of a subclass axiom is Car $\sqsubseteq$ Vehicle, that asserts that Car is a sub-concept of Vehicle. In OWL, a subclass axiom between atom classes is also called *is-a* relationship. The axiom Car $\sqsubseteq$ $\exists$ hasPart.SteeringWheel states that every instance of a Car has a part which is a steering wheel. Equivalency of concepts ($A \equiv B$), which means equiextension of the instance sets for any interpretation, is an abbreviation for pairs of axioms of the form $\{A \sqsubseteq B,\ B \sqsubseteq A\}$.

- **ABox axioms** are Assertional Box statements defining relationships between individuals. For example, the axiom hasFriend(Peter Griffin, Glenn Quagmire) is an assertion axiom, defining that the individual Peter Griffin has as friend the individual Glenn Quagmire.

- **RBox axioms** are Relational Box statements defining relationships between roles. For example, the axiom $\mathsf{hasBrother} \sqsubseteq \mathsf{hasSibling}$ states that the hasBrother is a sub-property of the hasSibling property.

The type of axioms that can be expressed about a knowledge domain depend on the Description Logic language that is used and the expressivity it allows. The type of axiom that all Description Logic languages allow is the atomic subsumption; that is the subsumption between atomic classes (e.g. $\mathsf{Car} \sqsubseteq \mathsf{Vehicle}$).

DL Expressivity

The expressivity of a DL language defines the set of constructors that can be used in axioms expressed in that language. Some basic DL languages are:

1. The $\mathcal{EL}$ logic allows for concept intersections ($\sqcap$) and existential restrictions ($\exists$).
2. The $\mathcal{ALC}$ logic allows for complex concept negation ($\neg C$, where C is a complex class, e.g. $\mathsf{Bat} \sqcap \mathsf{Man}$), concept union ($\sqcup$), concept intersection ($\sqcap$), existential restrictions ($\exists$) and universal restrictions ($\forall$).

Other constructors whose combination defines logics of higher expressivity are:

- $\mathcal{F}$ **Functional properties**: For a given individual, a role can have only one value. For example, the property assertion hasMother(Stewie, Lois), where Functional(hasMother), describes the fact that the individual Stewie can have only one mother, Lois. Thus, if the individual Stewie has two values for hasMother, these two values would be inferred to be alternative names for the same individual.
- $\mathcal{H}$ **Role hierarchy**: It defines relationship between properties (e.g. subproperties). An example is $\mathsf{hasSister} \sqsubseteq \mathsf{hasSibling}$.
- $\mathcal{R}^+$ **Limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness and transitivity**. An example is hasAncestor being a transitive role; if hasAncestor{Lois, Pewterschmidt} and hasAncestor(Meg, Lois), then it is implied that hasAncestor(Meg, Pewterschmidt).

- $\mathcal{O}$ **Nominals**: They define classes that can have only one individual as instance. For example, the concept {Brian} is a concept with only one instance, the individual Brian.
- $\mathcal{I}$ **Inverse properties**: They define properties of different direction. For example, the assertion hasMother(Stewie, Lois) has as inverse $\mathsf{hasMother}^{-}$(Lois, Stewie).
- $\mathcal{N}$ **Cardinality restrictions:** They can be used to define a number of role successors and predecessors of individuals. For example, the $\geq$ 2 hasSibling is a concept that defines individuals that have at least 2 siblings while the $\leq$ 2 hasSibling defines individuals that have maximum 2 siblings.
- $\mathcal{Q}$ **Qualified cardinality restrictions:** They are a more expressive form of cardinality restrictions, allowing the specification of role fillers. For example, the concept $\geq$ 2 hasSibling.Male defines individuals that have at least two male siblings.

The combination of constructors can define the expressivity of a language. For example, the DL $\mathcal{SHIQ}$ is the $\mathcal{ALC}$ plus extended cardinality restrictions, and transitive and inverse roles. Abbreviations on the languages can be defined. For example, the $\mathcal{EL}^{++}$ is an alias for $\mathcal{ELRO}$ and it allows in additional for transitive roles and nominals. The expressivity of OWL 2 DL is $\mathcal{SROIQ}$; it is an extension of $\mathcal{SHIQ}$ and in addition it includes reflexive roles.

2.3 An Ontology Language for the Web (OWL)

The basic notions of OWL are:

- **Entities:** elements used to refer to objects from a field of interest. In OWL, entities are concepts (classes), object and data properties, individuals, annotation properties and simple datatypes.
- **Expressions:** combinations of entities to form complex descriptions from basic ones. Expressions are complex classes in DL.
- **Axioms:** the basic statements that form an OWL ontology.

OWL	**DL Symbol**	**Manchester Syntax**	**Functional Syntax**
someValuesFrom	$\exists$	hasPet **some** Dog	ObjectSomeValuesFrom(hasPet Dog)
allValuesFrom	$\forall$	hasChild **only** Boy	ObjectAllValuesFrom(hasChild Boy)
hasValue	$\ni$	hasCountryOfOrigin **value** England	ObjectHasValue(hasCountryOfOrigin England)
minCardinality	$\geq$	hasSibling min 2	ObjectMinCardinality(2 hasSibling)
Cardinality	$=$	hasSibling **exactly** 2	ObjectExactCardinality(2 hasSibling)
maxCardinality	$\leq$	hasSibling max 2	ObjectMaxCardinality(2 hasSibling)
intersectionOf	$\sqcap$	Bat **and** Man	ObjectIntersectionOf(Bat Man)
unionOf	$\sqcup$	Man or Woman	ObjectUnionOf(Man Woman)
complementOf	$\neg$	not Dog	ObjectComplementOf(Dog)
subClassOf	$\sqsubseteq$	Car *SubClassOf* Vehicle	SubClassOf(Car Vehicle)
equivalentTo	$\equiv$	Batman *EquivalentTo* Bat **and** Man	EquivalentClasses(BatMan ObjectIntersectionOf(Bat Man))

Table 2.2: Representation of OWL constructors and examples in Manchester and Functional syntax

Axioms are used to make statements about a knowledge domain, relating entities. As mentioned above, an ontology, from a computational point of view, is a set of axioms describing a domain of interest.

OWL can be expressed in many formats, as already mentioned above. According to the W3C Recommendation [W3C09], all ontology tools should support the RDF/XML format. A common human-readable syntax for OWL ontologies is the Manchester OWL Syntax; another syntax easier on human eyes is the Functional Syntax. In this thesis, mainly Manchester OWL Syntax and DL syntax will be used for the demonstration of examples and results. Table 2.2 shows examples of OWL constructors in the Manchester and Functional Syntax.

2.3.1 OWL Sublanguages

OWL 1 has three main sublanguages; **OWL-Lite**, **OWL-DL** and **OWL-Full**. OWL 2 has the following profiles: **EL**, **QL**, **RL**, and OWL 2 DL, whose expressivity (SROIQ) is slightly higher than OWL DL (SHIQ) [MGH$^+$09]. OWL-Lite is the least expressive sublanguage of OWL. OWL-DL falls between that of OWL-Lite and OWL-Full. OWL-DL may be considered as an extension of OWL-Lite and OWL-Full an extension of OWL-DL. OWL-Full is the most expressive OWL variation and is undecidable, meaning that the reasoner cannot give a complete answer. Thus, performing automated reasoning on OWL-Full ontologies is not possible. OWL-Full is used in ontologies where high expressivity is more important than decidability and computational completeness of the language.

This thesis focuses on ontologies that use the OWL-Lite and OWL-DL languages.

2.3.2 Structural Notions and Graphical Representation

Following very closely the definitions described above for DL languages, each OWL axiom has an internal structure, which mainly connects entities with OWL constructs. Entities can be either classes, properties or individuals. Classes represent a set of individuals with common characteristics. Individuals are instances of classes and properties link pairs of individuals in relationships.

Signature of an ontology: It is defined as the set of atom entities in an ontology. These are the classes, object and data properties and individuals that exist in the ontology. The representation $Sig(\mathcal{O})$ denotes the signature of an ontology $\mathcal{O}$.

Usage axioms: The usage axioms of an entity are the set of axioms that reference that entity. For example given the axioms of Figure 2.2 the usage axioms for Car are axioms (1)-(3) while usage of property drives are axioms (3) and (4).

(1) Car *SubClassOf* Vehicle
(2) Car *SubClassOf* hasColor **some** Color
(3) CarDriver *SubClassOf* drives **some** Car
(4) LorryDriver *SubClassOf* drives **some** Lorry

Figure 2.2: Four example axioms describing cars and drivers.

Figures 2.3, 2.4 and 2.5 show different graphical representations of entities. Individuals are represented with diamond shapes. A set of individuals forming a class is enclosed in a circle. Finally, relationships between classes are represented with arrows connecting circles. Figure 2.5 shows the graphical representation of classes containing individuals. These individuals are related with properties (property assertions).

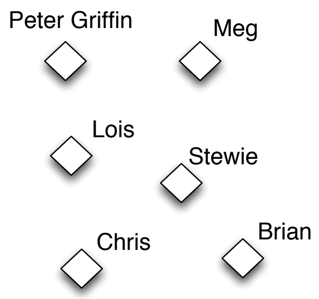


Figure 2.3: Representation of OWL Individuals.

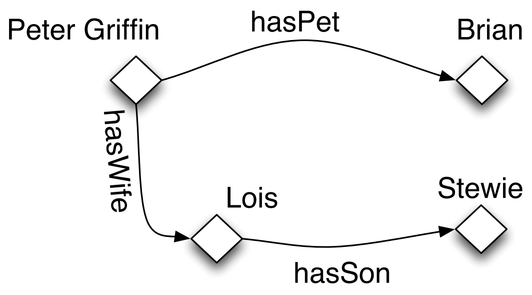


Figure 2.4: Representation of OWL Properties.

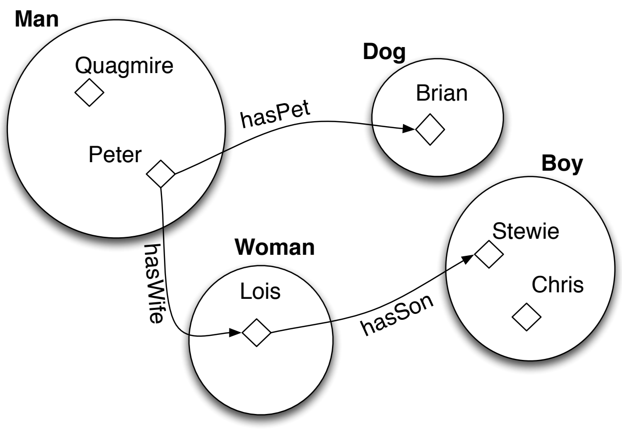


Figure 2.5: Representation of OWL Classes containing Individuals. The Individuals are connected with Properties.

In tutorials [HKR$^+$04], often the graphical representation of Figure 2.5 has been used to demonstrate the role of various objects in OWL. This kind of representation is based on set theory and it is regarded as an alternative, easier to understand, description of objects in OWL.

Is-a Relationship

Subclass axiom is also called an "is-a" relationship in OWL. Ontology editors usually represent such relationships as trees, although there is no single inheritance limitation in an OWL ontology. Two sample representations are shown in Figure 2.6 for the subclass axioms $\mathsf{Man} \sqsubseteq \mathsf{Adult}$, $\mathsf{Woman} \sqsubseteq \mathsf{Adult}$, $\mathsf{Boy} \sqsubseteq \mathsf{Child}$, $\mathsf{Girl} \sqsubseteq \mathsf{Child}$, $\mathsf{Child} \sqsubseteq \mathsf{Person}$, $\mathsf{Adult} \sqsubseteq \mathsf{Person}$.

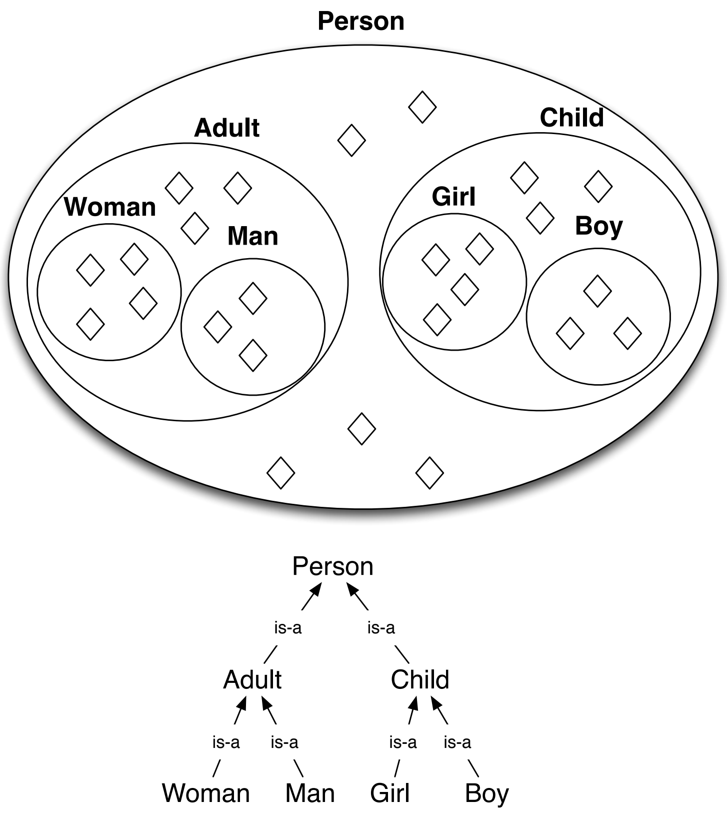


Figure 2.6: Two graphical representations of the class hierarchy. The first representation shows the subclass axioms as an overlap of the sets Person, Adult, Man, Woman, Child, Boy and Girl. The second representation shows the inheritance tree that is formed from the the subclass axioms.

Restrictions

Similarly, various graph representations can be used for existential and universal restrictions. Figure 2.7 shows two graphical representations of the existential restriction Father *SubClassOf* hasChild **some** Child. This axiom means that if there is an instance of the Father class, then it is connected to at least one instance of the Child class with the hasChild object property, even if the Child instance is not known. Secondly, the hasChild **some** Child is an *anonymous* class and it is a superclass of the Father class. This anonymous class can be also a superclass of the Mother class.

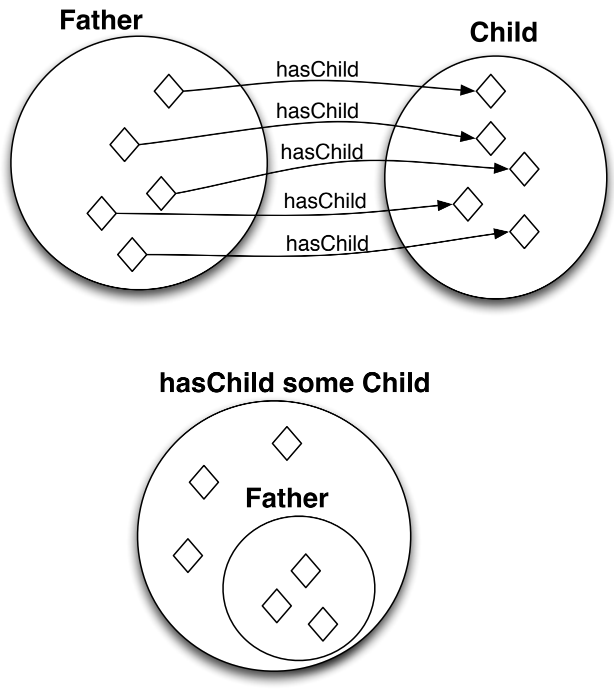


Figure 2.7: Two graphical representations of existential restrictions. The first representation shows how the individuals of two classes (Father, Child) are connected with the property (hasChild) of the existential restriction Father *SubClassOf* hasChild **some** Child. The second representation shows that the anonymous class hasChild **some** Child is a superclass of the Father class.

Universal restrictions can be represented similarly to the existential restrictions. The difference is that, for example, for the axiom LuckyFather *SubClassOf* hasChild **only** Girl, if there is an instance of a LuckyFather then all of its hasChild successors are inferred to be instances of the Girl class.

2.4 Reasoning

The power of OWL ontologies lies in the ability to perform sound and complete automated reasoning. Using automated reasoning the following tasks can be performed:

Consistency checking: Given an ontology $\mathcal{O}$ as an input, $\mathcal{O}$ is consistent if there is at least one model $\mathcal{I}$ of $\mathcal{O}$. In practice, this means that the reasoner checks if there are any contradictions in the ontology, thus checking whether the ontology is consistent. An inconsistent ontology has no models, thus, there is no interpretation that satisfies every axiom of the ontology. Consistency checking is the main standard reasoning service. Usually, consistency checking is performed before any other reasoning task.

Satisfiability Checking: Given an ontology $\mathcal{O}$ and a class expression C, C is satisfiable with respect to $\mathcal{O}$ if $\mathcal{O} \not\models C \sqsubseteq \bot$ i.e, $C^{\mathcal{I}} \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{O}$. In other words, an unsatisfiable class expression cannot have any instances in any model of the ontology, thus it is interpreted as an empty set in every model of the ontology [HBPS08], and flagged as a subclass of $\bot$, which in OWL is indicated with owl:Nothing (an unsatisfiable class is actually equivalent to $\bot$, since $\bot$ is also a subclass of any class, just as, in set theory, the empty set is a subset of any set).

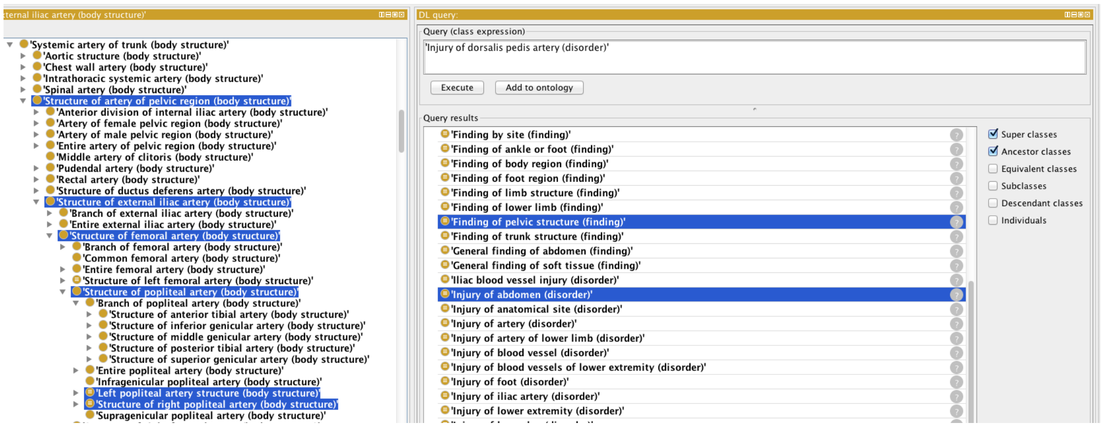
Subsumption Testing: Given an ontology $\mathcal{O}$ and two class expressions C and D, then is C subsumed by D ($C \sqsubseteq D$) with respect to $\mathcal{O}$ ($\mathcal{O} \models C \sqsubseteq D$)?; i.e., is $C^I \subseteq D^I$ in every model $\mathcal{I}$ of $\mathcal{O}$?. The reasoner infers the structure of "is-a" (subsumption) relationships, even if not all of them are explicitly stated. For example, if we have the following axioms in $\mathcal{O}$ $\{\mathsf{Man} \sqsubseteq \mathsf{Person}, \mathsf{Father} \sqsubseteq \mathsf{Man}\}$, then $\mathcal{O} \models \mathsf{Father} \sqsubseteq \mathsf{Person}$.

Instance Checking: Given an ontology $\mathcal{O}$, a class expression C and an individual α, the reasoner can infer if the individual α is of type C. In DL notation the entailment which is checked is $\mathcal{O} \models C(\alpha)$, i.e., if $a^I \in C^I$ for every model $\mathcal{I}$ of $\mathcal{O}$. For example, given the following axioms:

```
Mother EquivalentTo Woman and hasChild some Child
Individual: Lois
    Types: Mother
```

it is entailed that Lois is also an instance of Woman.

DL queries: Queries can be asked for getting answers on relations between entities. The reasoner will return all the results which are entailed by this query. DL queries can be used for getting answers for the knowledge represented in ontologies. The answers can be named superclasses (and ascendants), equivalent classes, subclasses (and descendant classes), or individuals, depending on the query. An example query is shown in Figure 2.8 as it is taken from DL query interface in Protégé 4.2. DL queries can be helpful for gaining an intuition of how an ontology works. For example, the Figure 2.8 shows a query for finding injuries of the foot ('Injury of dorsalis pedis artery (disorder)'). On the query results we have selected to show all the ancestor classes. Among the results we get injuries of the pelvis (e.g. Injury of the abdomen (disorder)), which was obviously not expected to be found. DL queries have been used for checking such modelling errors in the quality assurance of an ontology [RBS11].


```
xternal iliac artery (body structure)'
▼ 'Systemic artery of trunk (body structure)'
  ▶ 'Aortic structure (body structure)'
  ▶ 'Chest wall artery (body structure)'
  ▶ 'Intrathoracic systemic artery (body structure)'
  ▶ 'Spinal artery (body structure)'
  ▼ 'Structure of artery of pelvic region (body structure)'
    ▶ 'Anterior division of internal iliac artery (body structure)'
    ▶ 'Artery of female pelvic region (body structure)'
    ▶ 'Artery of male pelvic region (body structure)'
    ▶ 'Entire artery of pelvic region (body structure)'
      'Middle artery of clitoris (body structure)'
    ▶ 'Pudendal artery (body structure)'
    ▶ 'Rectal artery (body structure)'
    ▶ 'Structure of ductus deferens artery (body structure)'
    ▼ 'Structure of external iliac artery (body structure)'
      ▶ 'Branch of external iliac artery (body structure)'
      ▶ 'Entire external iliac artery (body structure)'
      ▼ 'Structure of femoral artery (body structure)'
        ▶ 'Branch of femoral artery (body structure)'
          'Common femoral artery (body structure)'
        ▶ 'Entire femoral artery (body structure)'
        ▶ 'Structure of left femoral artery (body structure)'
        ▼ 'Structure of popliteal artery (body structure)'
          ▼ 'Branch of popliteal artery (body structure)'
            ▶ 'Structure of anterior tibial artery (body structure)'
            ▶ 'Structure of inferior genicular artery (body structure)'
            ▶ 'Structure of middle genicular artery (body structure)'
            ▶ 'Structure of posterior tibial artery (body structure)'
            ▶ 'Structure of superior genicular artery (body structure)'
          ▶ 'Entire popliteal artery (body structure)'
            'Infragenicular popliteal artery (body structure)'
          ▶ 'Left popliteal artery structure (body structure)'
          ▶ 'Structure of right popliteal artery (body structure)'
            'Supragenicular popliteal artery (body structure)'
```

```
DL query:
Query (class expression)
'Injury of dorsalis pedis artery (disorder)'
[Execute] [Add to ontology]

Query results
'Finding by site (finding)'                               [✓] Super classes
'Finding of ankle or foot (finding)'                      [✓] Ancestor classes
'Finding of body region (finding)'                        [ ] Equivalent classes
'Finding of foot region (finding)'                        [ ] Subclasses
'Finding of limb structure (finding)'                     [ ] Descendant classes
'Finding of lower limb (finding)'                         [ ] Individuals
'Finding of pelvic structure (finding)'
'Finding of trunk structure (finding)'
'General finding of abdomen (finding)'
'General finding of soft tissue (finding)'
'Iliac blood vessel injury (disorder)'
'Injury of abdomen (disorder)'
'Injury of anatomical site (disorder)'
'Injury of artery (disorder)'
'Injury of artery of lower limb (disorder)'
'Injury of blood vessel (disorder)'
'Injury of blood vessels of lower extremity (disorder)'
'Injury of foot (disorder)'
'Injury of iliac artery (disorder)'
'Injury of lower extremity (disorder)'
```

Figure 2.8: An example of a DL query as it is shown in Protégé 4.

2.4.1 Tableaux Reasoners

The most common type of reasoners performing automated reasoning with DL systems are tableaux reasoners [BCM+03]. Tableaux algorithms try to prove the satisfiability of a concept C by constructing a model, an interpretation $\mathcal{I}$ in which $C^{\mathcal{I}}$ is not empty. A *tableaux* is a graph which represents interpretation models, with nodes corresponding to individuals (elements of the domain interpretation $\cdot^{\mathcal{I}}$). Thus, in order to check the consistency of an ontology, a tableaux reasoner attempts to build an interpretation model of this ontology.

The key idea of tableaux reasoners is that they check the consistency of an ontology by starting from an initial ABox of the ontology $\mathcal{A}_0$ and applying a set of expansion rules to break down $\mathcal{A}_0$; thus the representation of models in the ontology are represented as a set of ABoxes $\{\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n\}$. In languages such as $\mathcal{ALC}$ and more expressive ones, where rule expansion is non-deterministic, each ABox has decision points for the algorithm, thus each ABox has assertions which are the constraints on the associated model. For example checking the satisfiability of expression $\mathsf{C} \sqcup \mathsf{D}$ (x) will result in two ABoxes and therefore two models; one for C (x) and one for D (x). Each decision shows whether x is an instance of C or D. Table 2.3 shows the rules for $\mathcal{ALC}$ [BCM+03].

In this thesis, tableaux reasoners such as FaCT++ [TH06a], HermiT [SMH08] and Pellet [SPG+07] can be used in the discovery of semantic regularities; more details will be provided in the following chapters.

Rule	**Condition and Actions**
$\sqcap$-rule	If $(C \sqcap D)(x) \in \mathcal{A}$ and $\{C(x), D(x)\} \nsubseteq \mathcal{A}$ then $\mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$
$\sqcup$-rule	If $(C \sqcup D)(x) \in \mathcal{A}$ and $\{C(x), D(x)\} \nsubseteq \mathcal{A}$ then $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}, \mathcal{A}'' = \mathcal{A} \cup \{D(x)\}$
$\exists$-rule	If $(\exists R.C)(x) \in \mathcal{A}$ and $\{R(x,y), D(y)\} \nsubseteq \mathcal{A}$ for some y then $\mathcal{A}' = \mathcal{A} \cup \{R(x,y), C(y)\}$
$\forall$-rule	If $(\forall R.C)(x) \in \mathcal{A}$ and $R(x,y) \subseteq \mathcal{A}$ and $D(y) \nsubseteq \mathcal{A}$ for some y then $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$

Table 2.3: Expansion rules for $\mathcal{ALC}$

2.5 Modularity

Ontology modularity is an important research area of ontology engineering as it allows the partitioning and reuse of ontology fragments. In general, a module is a fragment of an ontology that is useful for some purpose. In practice, modularisation algorithms work as follows; the user provides a set of entities (signature) from the ontology which is used as a seed for extracting the set of axioms referring to this signature.

Two types of modularity are distinguished; the *syntactic* and the *semantic* one. Syntactic based approaches like the one presented in [SR07] consider only structural notions of axioms when extracting a module without considering the semantics of the language. These though do not guarantee the preservation of entailments from the original ontology. On the other hand, semantic based approaches [GHKS07a] produce modules that preserve the entailments that hold in the original ontology. Locality-based modules [GHKS07b] belong to this category; a module $M \subseteq \mathcal{O}$ for some ontology $\mathcal{O}$ and signature $\sigma \subseteq \mathsf{sig}(\mathcal{O})$ is guaranteed to preserve all entailments α that can be built from entities in σ which hold in $\mathcal{O}$.

Modularity is the basis for Atomic Decomposition, and for practical optimisations used in the experiments to deal with large and complex ontologies, in order to allow the reasoner to tackle such large ontologies in smaller portions. More details will be presented in the following chapters.

2.5.1 Atomic Decomposition (AD)

The Atomic Decomposition (AD) [VPS11] is a recent method that reveals the modular structure of an ontology. A naive approach to the computation of all modules in an ontology needs exponential time, since modules are generated depending on a subset of the entities mentioned in an ontology, and there is an exponential number of such subsets. The AD efficiently builds a structure that synthetically represents all modules based on locality for an ontology, regardless of the modularisation algorithm variant adopted and without the need to compute all modules in the ontology.

According to [VPS11], an atom is the smallest building block in the atomic decomposition structure, and consists of axioms co-occurring in modules. We use the following definitions from [Tsa12].

Definition 1 (Atom)**.** *Given an ontology $\mathcal{O}$ and a notion x of locality $\in \{\top, \bot\}$, an atom a is a maximal set of axioms, such that $\forall M = x\text{-}mod(\Sigma, \mathcal{O})$, we have either all of the axioms are in M or none of them does.*

In practice, the locality notion defines the way that a module extraction algorithm will constraint the ontology for the extraction of the atoms. In the $\bot$ locality, the modular extraction algorithm will constraint the ontology by excluding sub-concepts of a concept, while in the $\top$, the algorithm will extract the module by excluding super-concepts of a concept. In practice the locality type that is most commonly used is the $\bot$. In the remaining thesis when the type of locality is not mentioned, it will be implied the $\bot$-locality.

Definition 2 (Atomic Dependency)**.** *An atom a is dependent on an atom b (written as $b \preceq a$) if for all modules M such that $M \supseteq a$, then $M \supseteq b$.*

Definition 3 (Atomic Decomposition)**.** *The Atomic Decomposition AD of an ontology $\mathcal{O}$, is a dependency graph $G = \langle S, \preceq \rangle$, where S is the set of all atoms of $\mathcal{O}$.*

Since every atom is a set of axioms, and atoms are pairwise disjoint, the AD is a partition of the ontology, and its size is at most linear w.r.t. the size of the ontology [VPS12]. An axiom belongs to only one atom.

Lets consider the example ontology of Figure 2.9 taken from [VGK+11] and its $\perp$-AD shown in Figure 2.10.

(a_1) $\mathsf{Animal} \sqsubseteq (= 1\mathsf{hasGender}.\top)$,
(a_2) $\mathsf{Animal} \sqsubseteq (\geq 1\mathsf{hasHabitat}.\top)$,
(a_3) $\mathsf{Person} \sqsubseteq \mathsf{Animal}$,
(a_4) $\mathsf{Vegan} \equiv \mathsf{Person} \sqcap \forall \mathsf{eats}.(\mathsf{Vegetable} \sqcup \mathsf{Mushroom})$,
(a_5) $\mathsf{TeaTotaller} \equiv \mathsf{Person} \sqcap \forall \mathsf{drinks}.\mathsf{NonAlcoholicThing}$,
(a_6) $\mathsf{Student} \sqsubseteq \mathsf{Person} \sqcap \exists \mathsf{hasHabitat}.\mathsf{University}$,
(a_7) $\mathsf{GraduateStudent} \equiv \mathsf{Student} \sqcap \exists \mathsf{hasDegree}.\{\mathsf{BA}, \mathsf{BS}\}$

Figure 2.9: An example ontology consisting of 7 axioms [VGK+11]

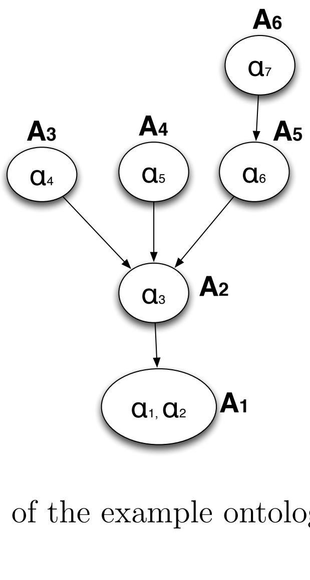


Figure 2.10: AD of the example ontology of Figure 2.9

The AD of Figure 2.10 consists of six *atoms* (A_1–A_6). Each atom holds a number of axioms. The AD graph also represents the partition of the ontology. For example, the module for atom A_6 will include the axioms in the path dependency from A_6 to A_1. Thus, the module M will include axioms $M = \{a_1, a_2, a_3, a_6, a_7\}$. Similarly the module for atom A_2 will include the axioms of atoms A_2, A_1.

The AD structure can have many useful applications such as quick extraction of modules [VGK+11], or to investigate dependencies between modules [VPS10], and finally improve reasoner performance for big ontologies [TP12].

As AD has the effect of reducing the number of axioms that need to be taken into account by a reasoner, it also helps in solving similar issues for humans; i.e., ontology debugging, mentioned above, can benefit from AD to reduce the number of candidate solutions when looking for ways to fix an inconsistent ontology; ontology comprehension is similarly simplified, as a smaller part of the ontology needs to be understood at once.

Even though the AD is a fine grained view of an ontology, it can provide information on the construction of the ontology and can explain how a module with a very small seed signature can lead to the extraction of a significant portion of the ontology. Also, the AD also guarantees that a module built on top of it will preserve all the entailments over the signature of the atoms, i.e., it will respect the notion of locality needed to guarantee its logical properties. This derives from the AD structure being based on the locality notion.

There is experimental evidence that the properties of the AD dependency graph can help to reveal logical dependency between axioms [VPS12]. In this thesis we use the AD structure for evaluating the discovered patterns, or regularities.

2.6 Ontology Comprehension

Ontology comprehension can be a demanding cognitive procedure. A user has to understand how OWL constructors are used to model the domain of the ontology and the implications they cause. The user has then to identify those parts of the ontology that have been modelled with the same combination or pattern of constructors and entities, together with their interconnection with the other parts of the ontology. In this way, the user is cognitively creating clusters of concepts with similar patterns. Therefore, the understanding of the pattern will consequently lead to the understanding of the cluster of concepts that has been designed using this pattern. Axiomatically rich ontologies make use of advanced characteristics of OWL leading to inferences that give the ontology a rich, interconnected and complex structure [RDH+04]. Consequently, a user can be faced with a logical artifact that is difficult to comprehend at many levels: The style of modelling; the size and shape of the whole ontology; the entailments afforded by the ontology; and so on. Even before the OWL language specification, ontologies sometimes failed to deliver their purpose as they were considered to be complex entities that required an excessive amount of time and effort to be used in conjunction with other ontologies or to be integrated into an application [UHW+98].

To the best of our knowledge, there is no existing framework for ontology comprehension. The term seems to have been introduced in [GWS07], where ontology comprehension was described as a problem that derives from the problematic "communication" between human agents and ontology artifacts. According to the authors of [GWS07], the *ontology comprehension* was described as:

the interaction between human agents and knowledge expressed in an ontology.

The authors distinguished two modes where ontology comprehension was important; the *development mode* and the *inspection mode*. In the development mode, comprehension of both the knowledge domain and its OWL description is important for the correct coding of the information. In the inspection mode, the understanding of the structure of the ontology, its underlying structures and design style are important factors for determining the reuse of an ontology.

Another effort on ontology comprehension is reported in [Kee07]. The author of this work refers to abstractions over an ontology and defines various layers of abstraction for facilitating the comprehension procedure. These abstractions are mainly conceptual abstractions defined by functions. Even though this work refers to abstractions, which are a key concept in comprehension, the abstractions that are defined are more ad-hoc solutions for particular ontologies that have similar structure by reusing upper ontologies like DOLCE [GGM+02].

In addition, another method developed for ontology comprehension is described in [BSP09]. The tool named as SuperModel aimed to provide an insight into the set of possible models for an ontology based on the implementation of a tableaux reasoner. The particular tool was coupled with the use of FaCT++ [TH06a] reasoner. The user could explore various models as these were exposed by the tool. A screenshot of the plugin for the Protégé 4 ontology editor is shown in Figure 2.11, taken from [HBPS08].

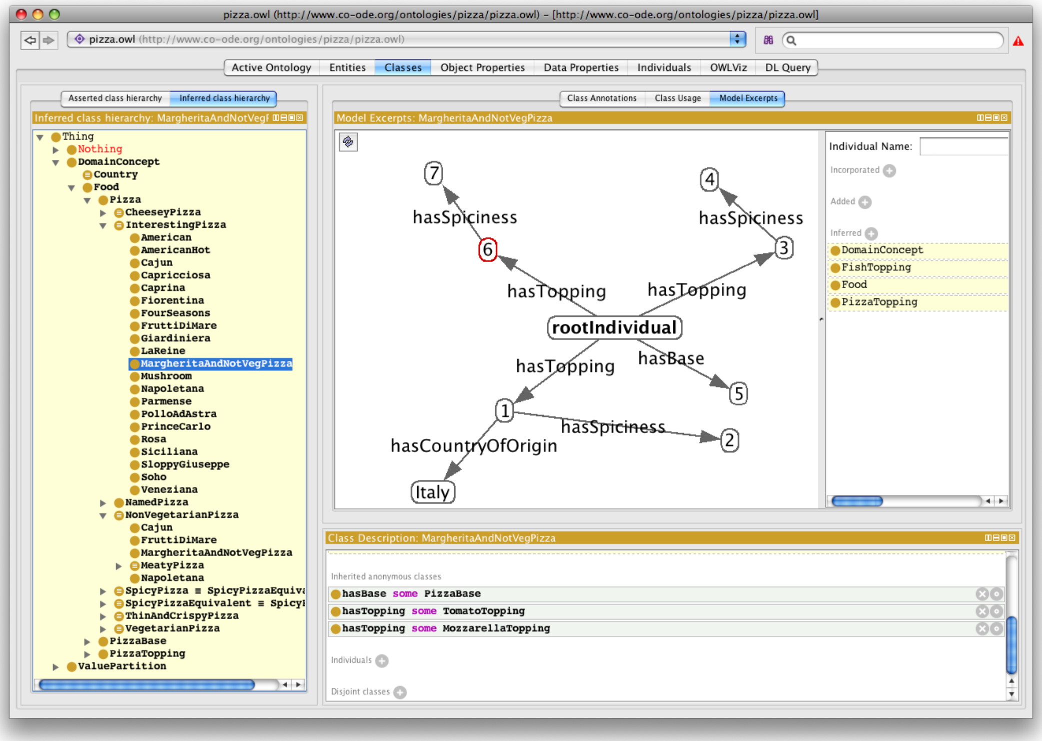


Figure 2.11: A screenshot of the SuperModel plugin in Protégé 4 [HBPS08]

2.6.1 Software Comprehension Analogy

The comprehension of an ontology can be compared to the problem of comprehending code. Even though there are many differences between code and OWL ontologies, structural organisation of information and artifact complexity are similar aspects.

Since there is no ontology comprehension framework available, we will use the analogy with software comprehension as an attempt to pin down corresponding aspects of ontology comprehension. Software comprehension [VML02] is the process of understanding code, its structure, organisation and purpose, in software engineering. The comprehension problem is derived from the concept assignment problem [BMW93] that is described in software engineering as the main problem of program understanding. The following notions are used in the majority of the presented software comprehension frameworks:

Reverse engineering according to [RS02] is the analysis of a subject system for identifying the system's components and their interrelationships, creating representations of a system in another form at a higher level of abstraction and understanding the program execution and the sequence in which it occurred.

Program slicing is a program reduction technique, allowing to narrow down the size of the source code of interest by identifying only those parts of the original program that are relevant to the computation of a particular function/output [RS02].

Dynamic slicing is a refinement of the static slice by only preserving the behaviour for a specific program input rather than all possible program input [Kor02].

Most notable software comprehension frameworks offer a variety of methods and tools for achieving code comprehension. The main points that are used in all of them is the provision of different abstraction layers. These tools mainly cover different aspects of code analysis such as low level slicing of the code or high level abstraction graphs for gaining an overview of the code's structure. Software comprehension frameworks like the DESIRE system [BMW93] or the MOOSE framework [RS02, NDG05, RK02] or the VizzAnalyser framework [LEL$^+$03, LP05] provide views and analyser tools of different code abstraction.

The functionality and features of the tools can be summarised to the following:

- Analysis tools performing code *slicing*. These tools mainly refer to the program slicing and dynamic slicing mentioned above and their goal is to narrow down the size of the code and isolate only the part that is of interest.

- *Visualisation techniques* for providing different views based on abstractions that are performed in the code. Abstract syntax trees, UML diagrams etc can be formed according to the structure of the code. For example, the VizzAnalyser framework [LEL$^+$03, LP05] uses metaphors for a more comprehensible visualisation of the software system. It provides the user with the flexibility to choose among a number of different metaphors and layout algorithms. With the use of metaphors, a program can be visualised as a city with buildings and streets. So a small building represents a small class or buildings on fire represent classes with bad code quality (e.g. without comments). Other visualisations refer to low level or high level analysis of the code.

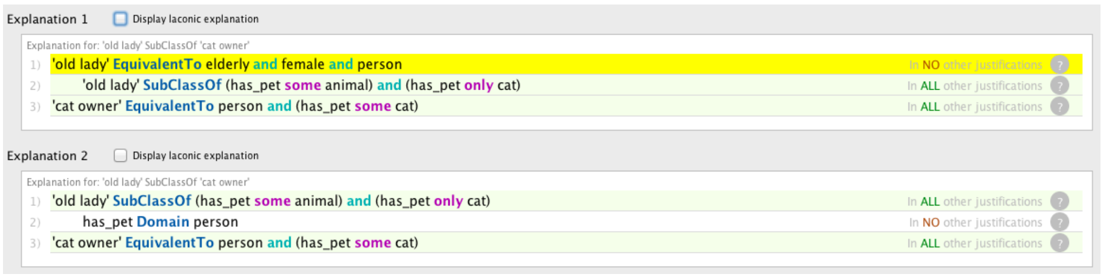
- *Cliché recognition methods* which identify repetitive structures in the code. In [Bro81] cliché recognition will result in *clusters* of similar code which can be browsed by the user.

Thus the reverse engineering methods that are provided by the software comprehension frameworks aim to provide abstractions for reducing the code complexity. Task oriented methods like program slicing aim to guide the user to achieve a specific inspection task. Visualisations are combined with the other techniques for the provision of more comprehensible abstractions over the code. The analogy with software comprehension can be helpful for the design of similar methodologies in the ontology comprehension area.

2.6.2 Aspects of Ontology Comprehension

By using the software comprehension analogy we can draw an outline of an ontology comprehension framework consisting of reverse engineering methods. This section describes such methods by reference to existing work. Most of these existing methods were already introduced in the beginning of this Chapter (Section 2.1) as solutions to other ontology engineering problems. However, we argue that these can be also used in the scope of reverse engineering.

Methods providing further analysis of the axioms work similarly to code slicing tools used in software comprehension frameworks. These are *modularisation* methods and methods used for *ontology debugging*, such as the provision of justifications for entailments. Modularity can help the user to extract the part of the ontology that is of interest and facilitate the analysis of an interconnected set of axioms. Justifications are explanations to entailments. These explanations are small fragments from the ontology that justify why an entailment hold. Most notable efforts on ontology debugging and on justifications are reported in [PSK05, HBPS08, HPS09a, HPS10b, KPHS07]. Two example justifications are shown in Figure 2.12, explaining why the entailment 'old lady' $\sqsubseteq$ 'cat owner' holds in the ontology. Justifications can be particularly helpful when inspecting an ontology and trying to understand the inferences made in the ontology. In the example of Figure 2.12, both explanations consist of three axioms. These axioms are asserted in the ontology. If one of the asserted axioms from the justification is removed, then the entailment will no longer hold. Justifications have been used for tracing unsatisfiable entities and inconsistencies in an ontology.


Explanation 1 ☐ Display laconic explanation

Explanation for: 'old lady' SubClassOf 'cat owner'

1) 'old lady' EquivalentTo elderly and female and person — In NO other justifications
2) 'old lady' SubClassOf (has_pet some animal) and (has_pet only cat) — In ALL other justifications
3) 'cat owner' EquivalentTo person and (has_pet some cat) — In ALL other justifications

Explanation 2 ☐ Display laconic explanation

Explanation for: 'old lady' SubClassOf 'cat owner'

1) 'old lady' SubClassOf (has_pet some animal) and (has_pet only cat) — In ALL other justifications
2) has_pet Domain person — In NO other justifications
3) 'cat owner' EquivalentTo person and (has_pet some cat) — In ALL other justifications

Figure 2.12: Two example justifications for the entailment 'old lady' $\sqsubseteq$ 'cat owner' as they are presented in Protégé 4.

Other methods that can be used in the scope of ontology comprehension are the abstractions provided by the Atomic Decomposition. The atoms of the dependency graph many times can consist of more than one axiom (see for Example Figure 2.10), thus are fine grained abstractions over the axioms of the ontology. On a second level the interconnection of atoms in the graph can help the user to trace dependent atoms.

Abstraction methods are closely related with efficient *visualisation approaches*. An analysis on ontology visualisation is reported in [TH06b]. Popular ontology editors like Protégé 4, Swoop, NeoN toolkit and TopBraid composer provide visualisations, simple ontology metrics and other features, such as property usage, for the structures of the ontology to facilitate comprehension. One important reason for using visual languages for knowledge representation is that an ontology might contain relationships that cannot be easily understood in pure text or formal logic. It is often easier to represent complex relationships with a picture that can be understood on an intuitive level. Protégé 4, on top of the basic hierarchy views, has plugins, such as OWLViz[11], for providing an alternative visual representation of the class hierarchy. It also allows for side-by-side viewing of asserted and inferred class hierarchies; a convenient view for assessing the ontology before and after reasoning. In general visualisation approaches are an important aspect for achieving comprehension.

Finally, methods on the *generation of natural language* from ontologies [Sch09] and methods analysing the differences between versions of an ontology [GPS11, GPS12] can contribute to a framework for systematic inspection and development of ontologies.

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

Cliché recognition is a common approach for *pattern detection* in software comprehension; such methods aim to highlight repetitive structures in the code and group parts of the code that have been developed in a similar way. However similar methods are missing in the landscape of ontology engineering. The existence of repetitive structures in an ontology are very closely related with the existence of ontology patterns. That is because patterns by definition impose a repetitive design over the data. Even though there are efforts on the development of ontologies using patterns there are not any methods for detecting patterns in ontologies. The next section gives an introduction to existing work on patterns and then presents the scope of this thesis.

2.7 Ontology Patterns

As it has been explained in Section 2.2, creating a formal ontology requires the creation of a conceptual vocabulary as well as the specification of relationships between the terms in that vocabulary; these relationships affect the conclusions, named as entailments, that are drawn from a given set of assertions. Capturing useful modeling decisions at a more abstract level of reusable *"patterns"*, i.e. representations which capture recurring structure within and across ontologies [Cla08] is an important aspect of ontology engineering and systematic ontology development.

In this section we will focus on two types of patterns; *knowledge patterns* and *ontology design patterns (ODPs).*

Patterns seek to raise the level of abstraction at which the ontology is formulated [IRS09]. This is achieved by instantiating a particular pattern of axioms that captures a particular modelling issue and enables the desired inferences to be made. Understanding may still be required, but the solution is ready-made and its reuse provides consistency of style [IRS09].

Knowledge Patterns, introduced in [Cla08] as:

> [...] representations which capture recurring structure within and across ontologies.

This notion, is similar to the concept from Software Engineering of Design Patterns [Pre94]. Similarly, the ontology community introduced the notion of Ontology Design Patterns (ODPs) [GP09, Gan05], which are best practices in modeling knowledge. According to the solution they offer, ODPs have been categorised into subcategories such as Logic and Content patterns [GP09]. The ODP portal[12] hosts a big variety of ODPs to facilitate their uptake. In the ODP catalogue, graphical explanation of a pattern is added and in most cases there is an implementation of the pattern in OWL. This is beneficial for inspecting the implementation of a pattern. The ODPs that are described in the catalogue are mainly best practices to specific problems. The benefit of developing ontologies by using design patterns and knowledge patterns, in general, is extensively described in [Ara09].

However, an unaddressed practical aspect is the instantiation of patterns in ontologies. These patterns can be either best practices (named as ODPs) or just design templates defined by the ontology developer. An ODP is a knowledge pattern, but the converse does not necessarily hold in general, i.e., a knowledge pattern might not be represented as an ODP [IRS09]. A problem that the OPPL scripted language aimed to address is the systematic instantiation of patterns in OWL ontologies [IPRS10, IERS08, IRS09].

2.7.1 Ontology Quality Assurance

The quality assurance of an ontology is closely related to the ontology comprehension problem and many efforts have been reported towards that direction. These methods mainly involve the definition of metrics that aim to assess the quality of an ontology in terms of its complexity, information richness, documentation etc. The methods are reported in [BJSSA05, YOE05, YZY06, MJF10, AB06].

Pattern based approaches for the quality assurance of particular ontologies like SNOMED-CT, FMA and and GO are described in [MRS09, FBIP$^+$10, RBS11, CSKD04]. These methods seek particular types of errors which are anchored to the semantics of the ontology. The methods that are adopted either involve only manual inspection of the ontology or they also include methods for analysing potential underlying patterns that can be mapped to the semantics of an ontology. A more systematic analysis is reported in [FBIP$^+$10] where the authors performed an analysis of the axiomatic description of terms with similar naming conventions in order to highlight patterns that were adopted for the design of these terms and axiomatically enrich them. In that work, the OPPL language was used for matching a pattern and for ingesting new ones.

[12]`http://ontologydesignpatterns.org/`

However, all these methods have in common the requirement of manual inspection of an ontology. This will be the starting point where the ontology engineer will manually inspect the description of a particular set of entities (e.g. entities with a particular keyword in their label) in the ontology. Based on the manual inspection the engineer will express possible patterns for this set of terms like the authors in [RI12]. This pattern will then be used as a query against the ontology with the use of a scripting language like OPPL which is described in detail in the following Section. By inspecting the instantiation of this pattern the engineer will be able then to form a hypothesis for special cases in the ontology; e.g. entities that were expected to instantiate the pattern but instead deviated from that pattern. However, there is a lack of methods to highlight patterns in an ontology for facilitating the above procedure. This thesis aims at dealing with this problem, as mentioned above.

2.7.2 OPPL

The Ontology Pre-Processor Language (OPPL)[13] is a language for querying and modifying Description Logic knowledge bases expressed in OWL [IERS08]. OPPL was initially motivated by the need of ontology developers to transform an ontology to an axiomatically richer one [ERSA08].

An OPPL statement has the following form:

```
?varible_1:Type ... ?varible_n:Type
SELECT Query,....,Query
BEGIN
ADD | REMOVE Axiom
...
ADD | REMOVE Axiom
END
```

Thus, the main parts of an OPPL statement are:

1. Variable declaration (`?variable_1:Type`)
2. Selection (between `SELECT` and `BEGIN`)
3. Actions (between `BEGIN` and `END`)

[13]`http://oppl2.sourceforge.net/`

The OPPL syntax is built upon the Manchester OWL Syntax [HPS09b]. The full OPPL syntax grammar is shown in Figure 2.13. An important part in the OPPL script is the variables. They are used in an OPPL script to represent a set of entities that play similar role in an axiom. Then, queries and actions are expressed using the variables, enabling batch processes on the axioms. For example the following OPPL script:

```
?target:CLASS, ?origin:CLASS
SELECT
?target SubClassOf develops_from some ?origin
BEGIN
ADD ?target SubClassOf develops_from only ?origin
END
```

defines two variables (`?target`, `?origin`) that hold classes. In this script the bindings for the variables will be all classes that participate in the general axiom

```
?target SubClassOf develops_from some ?origin
```

and for these bindings will add a universal restriction (`ADD ?target SubClassOf develops_from only ?origin`).

In general, variables can have the following types:

1. `CLASS;`
2. `OBJECTPROPERTY;`
3. `DATAPROPERTY;`
4. `INDIVIDUAL;`
5. `CONSTANT.`

Each variable type covers a possible category of entities in the OWL specification. An entity is a named object or a constant. Therefore, it should be noted that an anonymous class description is not an acceptable substitution for any variable type as this will affect the completeness of the underlying algorithms for executing arbitary queries [IRS09].

OPPL is can be used for embedding patterns in an OWL ontology [IRS09]. It can be used for specifying general knowledge patterns in OWL, whether these

```
OPPL Statement  ::=  ( VariableDeclaration )?
                     ( Query )? ( Actions )? ";"
VariableDeclaration  ::=  VariableDefinition
  ( "," VariableDefinition )*
Actions  ::=  "BEGIN" Action ( "," Action )* "END;"
VariableDefinition  ::=  <IDENTIFIER> ":" variableType
Constraint  ::=  <IDENTIFIER> <NEQ> OWLExpression
variableType  ::=  "CLASS" | "OBJECTPROPERTY |
"DATAPROPERTY" | "INDIVIDUAL" | "CONSTANT"
Query  ::=  "SELECT" Axiom ( "," Axiom )*
( <WHERE> Constraint ( <COMMA> Constraint )* )?
Action  ::=  "ADD" | "REMOVE" Axiom
Axiom  ::=  An axiom in Manchester OWL Syntax
(possibly containing variables)
IDENTIFIER::= "?"<LETTER> (<LETTER>|<DIGIT>)*
LETTER ::= ["_","a"-"z","A"-"Z"]
DIGIT ::= ["0"-"9"]
OWLExpression  ::=  An OWL entity in Manchester OWL Syntax
(possibly containing variables)
```

Figure 2.13: The OPPL syntax [IERS08]

are ODPs or not. It has the following advantages for expressing patterns in ontologies:

- Provision of a declarative specification of patterns and their descriptions in OWL.
- Automatic production of pattern instantiations. This can be helpful for facilitating automated population of ontologies based on templates [JHI+10].
- Ability of formal analysis of patterns and their effects of their reuse. This can be helpful for ontology engineer for comprehending fragments of the ontology that instantiate the pattern.
- Their specification as a reusable template for extending an ontology, offers a means of expressing the ways of expected evolution of an ontology with respect to the intentions of its original modelers.

Thus, OPPL can help the manipulation and modification of axioms in an ontology to instantiate a particular pattern. Another practical usage of OPPL demonstrated in [JKS+11, JHI+10] is the population of ontologies from spreadsheets. In particular in [JHI+10] the authors describe a usage scenario for developing an ontology which was deemed more efficient for biologists. A spreadsheet was used to collect the knowledge from biologists while ontology engineers extracted this knowledge from spreadsheets and populated an ontology with the use of OPPL.

However, an aspect that has not being addressed yet is the mining of patterns in ontologies when such an ontology is shared and reused from other people. To the best of our knowledge, at the time of writing this thesis, the only way to understand whether an ontology uses a pattern for the description of axioms is by manually inspecting the ontology. However, such a procedure becomes impractical especially for big ontologies.

2.7.3 Need for pattern detection approaches

Section 2.7 introduced various notions of patterns and described their usefulness for ontology development, reuse, maintenance and quality assurance. Thus, patterns play a key role in the systematic development and reuse of an ontology.

Existing approaches are focusing mainly on the development of ontologies with patterns. However, the verification of existence of patterns in an ontology is a process that has to be done manually by the ontology engineers who are reusing the ontology. If the patterns have not being described in the documentation of an ontology by the developers of the ontology, then the procedure of mining such patterns is a difficult task as it is done by manual inspection of axioms and formulation of OPPL queries and DL queries to verify a pattern.

This thesis tackles this problem with a machine learning approach for the unsupervised detection of patterns in an ontology. In particular, the research presented in this thesis deals with the detection of repetitive structures named as *regularities*, considering two aspects of an ontology; its asserted axioms and its entailments. The aim is to highlight repetitive structures in the asserted axioms and entailments of an ontology and express them with more generalised expressions. These generalised expressions could highlight the underlying patterns that were used for the design of similar parts of an ontology, and give an intuition of the construction of the ontology, as well as highlight deviations from the norm.

2.8 Summary

This chapter introduced the basic concepts of ontology engineering and highlighted the necessity for reverse engineering methods. It established the problem of ontology comprehension and presented the landscape of a framework for facilitating such a procedure as the understanding of an ontology. It also introduced background work for this thesis and the usefulness of patterns in ontology engineering. Patterns have been used for embedding knowledge in ontologies; these can be best practises or design decisions adopted by the developers of the ontology. Patterns can play a key role in the inspection of an ontology; having knowledge about them can help the user to understand the construction of an ontology. In the broad area of reverse ontology engineering we have drawn, we focus on a particular sub-problem, which is the unsupervised detection of patterns in ontologies. In particular, although there are methods and tools for embedding patterns in an ontology, there is a lack for the reverse procedure; mining patterns from an ontology. At the moment, such procedure is done manually. The unsupervised detection of regularities in the axioms and entailments of an ontology can organise the axioms and entailments according to their repetitive design style, thus can help the user to inspect the construction of an ontology. In this thesis, we are dealing with this problem of unsupervised detection of regularities. The following chapters give details on our approach to solve this problem.

Chapter 3

Regularity Inspector for Ontologies (RIO) Framework

Chapter 2 presented the current problems with the reuse of ontologies. As mentioned before, many ontology reusing scenarios share a need for *ontology comprehension*, meaning the understanding of an ontology. Even though the level of understanding that is needed varies depending on the task at hand, understanding underlying patterns and mapping the axiomatisation to the corresponding knowledge domain are almost always needed. In Chapter 2 we also presented the outline of an ontology comprehension framework for assessing the various dimensions of the problem.

The design of ontologies based on patterns is an advisable approach as this can improve the quality assurance process of the ontology and in general it can help the systematic development of the ontology; the reason for this is that, in an ontology that follows a set of patterns, irregularities are more evident and are likely to point to errors. In software engineering, the term “code smell” expresses a similar concept: constructs and patterns that are likely to be errors or that will make future development harder [FB99].

However, the reverse procedure, i.e., mining patterns from ontologies, has not beeing investigated yet. This thesis aims to contribute to this area with the establishment of a framework for the detection of regularities in ontologies. However, the methods described here cannot be considered as standalone methods for achieving comprehension. Ontology comprehension as a goal is not the scope of this thesis but the reverse engineering methods described here can be integrated with other systems for creating an ontology comprehension framework.

This chapter presents the architecture of the Regularity Inspector for Ontologies (RIO) framework. RIO, the main outcome of this thesis, performs unsupervised detection of syntactic and semantic regularities in ontologies.

This chapter presents an overview of the framework and describes the main architecture and algorithms for computing syntactic and semantic regularities. It introduces the challenges and the various implementation decisions that were taken in order to tackle these challenges, and it investigates potential use cases in which the detection of regularities can be used as an ancillary service. Chapters 4 and 5 give details for the computation of syntactic and semantic regularities respectively.

3.1 Ontology Regularities

This section presents the nomenclature and some basic definitions that are used throughout this thesis and makes distinctions between different types of patterns.

We distinguish the notion of syntactic regularity from the notion of *design pattern*, since the term pattern has multiple meanings. In ontology engineering, the existence of patterns is interpreted as design patterns, i.e., solutions to design problems [GP09, ERSA08, Gan05]. Patterns of axioms, however, can exist throughout an ontology without being an accepted design pattern. Design patterns can be represented in different forms, such as conceptual models, with OPPL scripts [IERS08, IRS09], text descriptions etc [PRG$^+$09]. However, a regularity is not necessarily a design pattern.

An axiom α is asserted in an ontology $\mathcal{O}$. We will omit the ontology $\mathcal{O}$ where it is clear from the context. Thus, we define the following:

Definition 4 (Generalisation function). *Let a be an axiom and $e \in sig(a)$ be an entity and $?\mathsf{x}$ be a variable. The generalisation function $g(a, e, ?\mathsf{x})$ is a function whose output is the axiom a where the entity e is replaced by the variable $?\mathsf{x}$.*

Definition 5 (Generalisation). *Let $n > 1$ be an integer, $A = \{a_1, \ldots, n\}$ be a set of n axioms such that there exists a set of entities $E = \{e_1, \ldots, e_n\}$ with $g(a_i, e_i, ?\mathsf{x}) = g(a_j, e_j, ?\mathsf{x})$ $\forall i, j \leq n$. The generalisation $G(A, E, ?\mathsf{x})$ over a set A for the entities in E is the axiom obtained through the generalisation function $g(a_i, e_i, ?\mathsf{x})$ applied to any axiom $a_i \in A$.*

Since, $G(A, E, ?\mathsf{x})$ is the same as $g(a_i, e_i, ?\mathsf{x})$ we will refer to a generalisation with $g(a_i, e_i, ?\mathsf{x})$ and for brevity we will use $g(a_i)$ or just g. We borrow the syntax for the variables in a generalisation from OPPL[1], a declarative language for manipulating OWL ontologies [IERS08]. An example generalisation is ?x *SubClassOf* Vehicle, where ?x is a class variable holding all classes that are vehicles.

Thus, a generalisation can express a *syntactic regularity* or *syntactic pattern*, as it can abstract over a set of axioms with the same structure. Similarly, a *semantic regularity* or *semantic pattern* can abstract over a set of entailments with same structure.

The term "semantic patterns" has been introduced in [SEM01] for describing high level solutions to a modeling problem. However, here, the intuition of a *semantic regularity (or pattern)* is that it represents implicit information made available by automated reasoning.

To give an example, consider the following axioms:

(1) Car $\sqsubseteq$ Vehicle
(2) Sports_car $\sqsubseteq$ Car
(3) Bus $\sqsubseteq$ Vehicle
(4) Mini_Bus $\sqsubseteq$ Bus

These four axioms are subclass axioms between atom classes. A type of syntactic regularity that can be detected in the previous axioms is

(a) ?x $\sqsubseteq$ Vehicle
(b) ?x $\sqsubseteq$ Vehicle
(c) ?y $\sqsubseteq$?x
where ?x={[Car, Bus], ?y=[Sports_car, Mini_Bus]}

Axioms (a),(b) and (c) can be regarded as *generalised axioms* that have variables holding similar entities. Being able to detect such structures can be useful to gain an insight to the construction of an ontology. The main contribution of this thesis is RIO, a framework that is able to detect such generalisations.

3.2 RIO usage scenarios

RIO mainly focuses on the detection of syntactic and semantic regularities. Pattern recognition is an integral part of most machine intelligence systems built for decision making [TK06].

[1]Details on OPPL are given in Section 2.6

The detection of syntactic regularities deals with the way axioms are asserted in an ontology; for example, it reveals general patterns that the ontology engineer has followed while writing the axioms that describe the domain of the ontology. This aspect can be potentially helpful for revealing the compositional style of the ontology, and, thus, facilitate the comprehension of an ontology.

Some use cases in which RIO can be helpful as a service are:

1. **Comprehension of an ontology:** This is a very general task in which the user is inspecting an existing ontology, which is later going to be used for completing a task; for example, integrating the ontology in the back-end of a system. In this task then the user has to inspect the ontology, understand how the domain of the ontology is described and be able to reuse parts of the ontology.

2. **Extension of an ontology:** This task mainly involves the process of inspecting an ontology with the goal of extending it. This requires the identification and "understanding" of patterns that are used for describing different parts of the ontology. Then, the adoption of an existing pattern for the description of new entities in the ontology will enable a uniform extension of the ontology with the expected behaviour. It will also facilitate the maintenance of the ontology, as the comprehension of a uniform design style will suffice.

3. **Quality assurance of an ontology:** The detection of regularities in an ontology can be helpful in the broader area of quality assurance of an ontology. One aspect of quality assurance is for checking conformance to intended patterns. This involves the inspection of patterns for understanding how the ontology was built and how a topic in the ontology domain is mapped to a pattern. Being able to analyse the composition of the ontology and to identify the underlying patterns can be helpful for distinguishing the different design styles that were followed during the development of the ontology. In this thesis, we focus mainly on this application of RIO.

4. **Analyse the construction of an ontology:** An ontology engineer wants to reuse an ontology for an application. The engineer has some prior information about the ontology and an intuition of some of the dominant patterns that are used in the ontology. An analysis of the regularities of the

ontology can be helpful for verifying the existence of these patterns and for showing which entities from the ontology instantiate these patterns.

5. **Analysis of the evolution of an ontology:** The implementation of such a scenario presupposes the coupling of RIO with a toolkit for analysing differences between ontologies [GPS12]. The authors in [GPS11] categorise types of differences in the ontology according to the impact they have on the entailments of the ontology. This analysis is applied for all the versions of an ontology and shows the evolution of an ontology with respect to its semantics. A framework for detecting regular structures in an ontology, like RIO, can be used for exploring the distribution of differences in an ontology to corresponding patterns. For example, such an analysis will give a better intuition on how the ontology was constructed; whether ontology developers focus on one or more patterns per version, how these patterns alter and deviate through different versions, and so on.

6. **Generation of artificial ontologies for benchmarking:** Benchmarking systems like Lehigh University Benchmark (LUBM) [GPH05] or the Hladik [Hla05] have used an approach of "artificial" generation of an ontology or in other words synthetic generation of schemas for testing reasoners. Further approaches to generating synthetic RDFS schemas and OWL ontologies "from scratch" use commonly occurring patterns and distribution laws, as described in [TGC07, Š07], amongst others. The idea is to extend the schema of an ontology and predict its evolution in terms of reasoning performance under the assumption that no new constructors will be introduced. This extension is a large scale "realistic" ontology. However, given the frequent occurrence of design patterns and structural regularities in OWL ontologies, a randomised synthetic ontology generation might lead to a far more diverse set of axioms than we expect to find in naturally occurring ontologies. The syntactic regularities that RIO detects could be used as a template for the ontology schema, which can be further populated for extending the ontology.

3.3 Architecture of RIO framework

RIO is a framework for the unsupervised detection of regularities in ontologies. In our definitions, we distinguish the notion of regularities from the notion of patterns in the following sense; regularities are on a lower level from patterns as they just express repetitive information in the ontology. A regularity is a set of axioms with reoccurring structure, while a pattern has been used with the meaning of a design pattern: best practice for a design problem. In the remaining chapters, referring either to patterns or regularities will have the same meaning. The general usage of the word pattern differs from the word “design pattern”.

The detection of regularities is focused on two aspects:

1. How was the ontology composed?
2. What are the most important inferences that can be drawn from the ontology? What can we ask about the ontology?

The first question is answered with the detection of *syntactic regularities* in an ontology while the second one can be addressed with the detection of *semantic regularities* in a selected set of entailments in the ontology. RIO incorporates algorithms for detecting repetitive structures either in the axioms or entailments of an ontology. These repetitive structures are expressed in the form of generalisations, which are axioms with variables replacing a set of entities. More formally, a generalisation is defined as follows:

The main question and corresponding ‘problem’ in the detection of regularities (repetitive structures) in an ontology is how we detect these repetitive structures. This question can be reformulated to the following question: What granularity level should we use for detecting the repetitive structures in an ontology? In other words, given a set of axioms, which entities do I select to represent with variables? This will have an impact on the way we represent the regularities, and consequently the corresponding patterns in the ontology and the level of abstraction at which we choose to represent them.

3.3.1 Detecting syntactic regularities

Given the axioms of Figure 3.1 taken from the Amino Acid ontology[2], syntactic regularities of different granularity can be expressed.

[2]`http://www.cs.man.ac.uk/~stevensr/ontology/amino-acid.owl`

```
(1) Alanine SubClassOf hasCharge some Positive
(2) Aspartate SubClassOf hasCharge some Negative
(3) Cysteine SubClassOf hasPolarity some Non-Polar
(4) Glutamate SubClassOf hasPolarity some Polar
```

Figure 3.1: Four axioms from the Amino Acid ontology, describing four amino acids.

For example, Figure 3.2a shows two example regularities, g $_1$ and g $_2$, where in g $_1$, it is ?AminoAcid =[Alanine, Aspartate] and ?Charge =[Positive, Negative] and in g $_2$, it is ?AminoAcid =[Cysteine, Glutamate], ?Polarity =[Non-Polar, Polar]. Thus, g1 *instantiates* axioms (1) and (2) and g2 instantiates axioms (3) and (4).

```
g1 = ?AminoAcid SubClassOf hasCharge some ?Charge
g2 = ?AminoAcid SubClassOf hasPolarity some ?Polarity
```

(a) Two example regularities abstracting the four axioms of Figure 3.1.

```
g1' = ?AminoAcid SubClassOf ?hasPhysicoChemicalProperty
                            some ?PhysicoChemicalProperty
```

(b) An example regularity abstracting the four axioms of Figure 3.1.

Figure 3.2: Two examples of regularities of different granularity, abstracting the four axioms of Figure 3.1.

However, a single regularity can also used to express the repetitive structure of the axioms in Figure 3.1. Such a regularity is shown in Figure 3.2b, where ?AminoAcid =[Alanine, Aspartate, Cysteine, Glutamate], ?hasPhysicoChemicalProperty =[hasCharge, hasPolarity] and ?PhysicoChemicalProperty =[Positive, Negative, Non-Polar, Polar].

In this example, all generalisations (g_1, g_2, g_1') expressing the underlying repetitive structures are correct and meaningful; entities that are represented by a variable are also similar in context (for example, variables holding all Amino Acids, or Polarity values etc). The only difference is that g1' is more abstract than g_1, g_2. A different example taken from the Pizza ontology[3] is shown in Figure 3.3a.

MargheritaPizza *SubClassOf* hasBase **some** PizzaBase
MargheritaPizza *SubClassOf* hasTopping **some** MozzarellaTopping
LaReine *SubClassOf* hasTopping **some** SpinachTopping
SloppyGiusepe *SubClassOf* hasTopping **some** TomatoTopping

(a) Four axioms taken from the Pizza ontology.

g_1'=?Pizza *SubClassOf* ?PizzaProperty **some** ?PizzaFiller

(b) A generalisation example abstracting the four axioms.

Figure 3.3: Four axioms taken from the Pizza ontology and an example regularity describing the four axioms.

A generalisation that describes these axioms is shown in Figure 3.3b where, ?Pizza =[MargheritaPizza,LaReine, SloppyGiusepe], ?PizzaProperty =[hasBase, hasTopping], ?PizzaFiller =[PizzaBase, MozzarelaTopping, SpinachTopping, TomatoTopping]. Such regularity seems to be very generic for entities that represent different "topics" in the ontology (E.g. PizzaBase, PizzaTopping).

Therefore, one of the main challenges is to decide which entities to replace with variables in a way that can capture the underlying semantics of the ontology (e.g. a variable that will hold entities that are also similar in context).

From a computational point of view, selecting which entities to replace with a variable is not an easy task. A naive solution would be trying all possible combinations of entities for selecting the variables. However, there are exponentially

[3]`http://www.co-ode.org/ontologies/pizza/pizza.owl`

many combinations to try and thus this solution is not practical.

On the other hand, machine learning, and clustering in particular, can help to overcome this computational obstacle. Cluster analysis can help to define groups of similar entities. These clusters of similar entities then can serve as variables when expressing the generalisations.

3.3.2 RIO Workflow

The workflow of RIO for the detection of regularities in an ontology is shown in Figure 3.4 and it is similar to the Knowledge Discovery process (KD) [Lan05]. The details of these blocks differ for the computation of semantic and syntactic regularities. For example, in Figure 3.4, the extraction of a finite set of entailments was done using the Knowledge Explorer (KE). Details about it are given in Chapter 5. The general workflow remains the same for the computation of both types of regularities.

In the remaining sections we will describe the general blocks in the workflow. These are:

1. Selection of the relevant ontology fragment for regularity detection.
2. Proximity matrix computation.
3. Cluster analysis.
4. Generalisations expression.

The computation of task 1 depends on the type of regularities we want to compute. For the computation of syntactic regularities, the relevant part is the set of asserted axioms in the ontology. For the computation of semantic regularities, the relevant part is the set of entailments extracted from the ontology.

Tasks 1 - 2 are the preprocess for clustering. For a more cohesive reading, we will start describing task 3 and then proceed with the description of task 2.

3.4 Cluster Analysis

Cluster analysis has been used in different disciplines for tackling the problem of unsupervised pattern recognition. The problem that *cluster analysis* addresses is defined according to [Lan05] as:

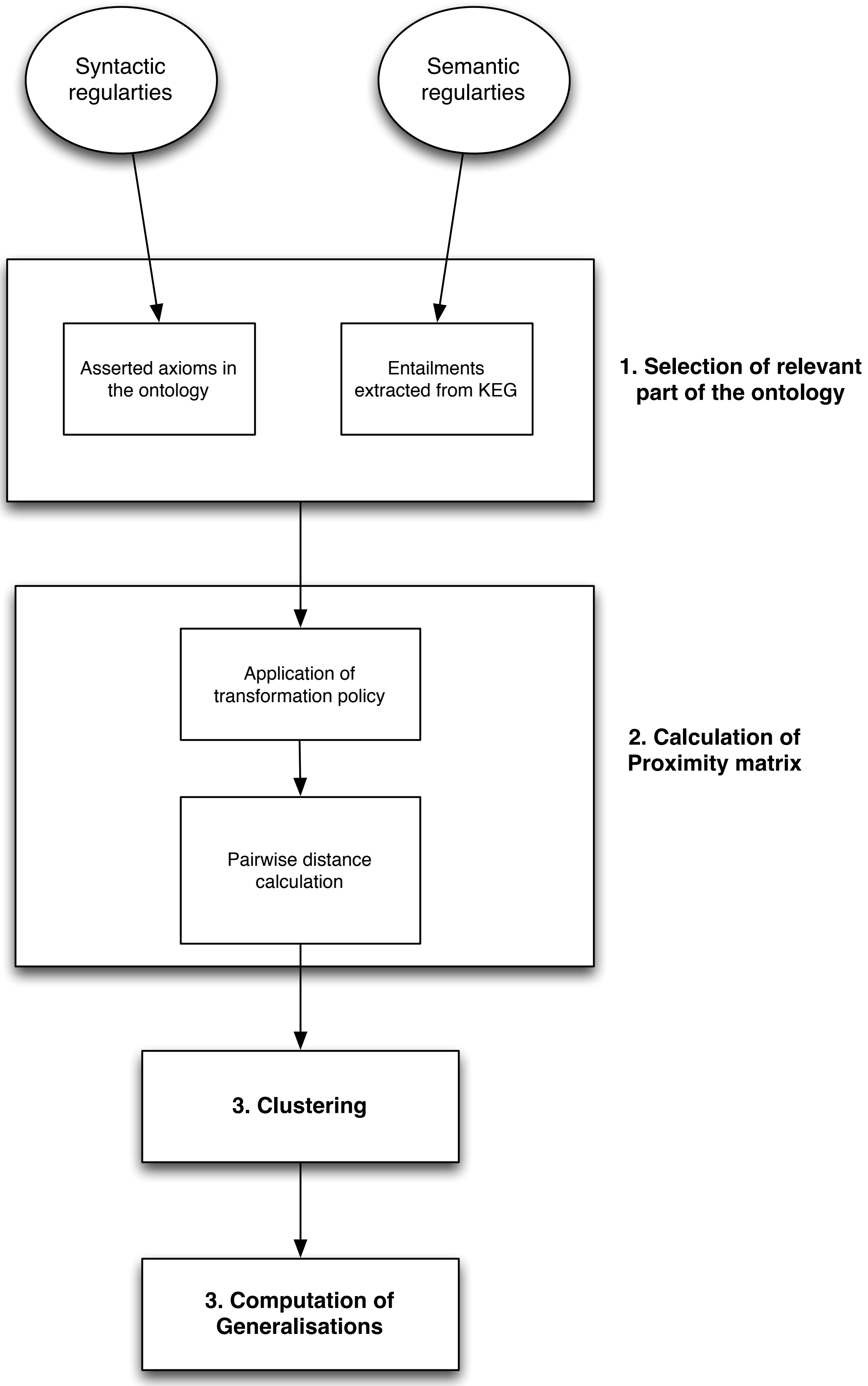

Syntactic regularties
Semantic regularties
Asserted axioms in the ontology
Entailments extracted from KEG
1. Selection of relevant part of the ontology
Application of transformation policy
Pairwise distance calculation
2. Calculation of Proximity matrix
3. Clustering
3. Computation of Generalisations

Figure 3.4: Workflow of RIO framework.

"Given a collection of n individual objects, each of which is described by a set of p characteristics or variables, derive a useful division into a number of classes. Both the number of classes and the properties of the classes are to be determined."

Thus, cluster analysis will give a partition of n objects, that is a set of clusters where an object belongs to *one* cluster only.

RIO framework uses cluster analysis in order to get clusters of similar entities, with similar usage in the axioms of an ontology. Even though different clustering algorithms can be used, in this thesis we focus on the results that we obtain from one kind of clustering rather than explore all the alternative clustering algorithms.

In the literature there are several clustering techniques; we chose agglomerative hierarchical clustering [Lan05] and we report an informal description of the algorithm we used in Algorithm 1. A more detailed description of the algorithm is given in Section 4.6.

Algorithm 1 Basic agglomerative hierarchical clustering

```
Input: Proximity matrix with distances
Output: A set of clusters
  Assign to the current set of clusters the set of singleton clusters, each repre-
  senting an input entity.
  repeat
     Merge the closest two clusters, replace them with the result of the merge in
     the current set of clusters.
     Update the proximity matrix, with the new distance between the newly
     created cluster and the original ones.
  until The stopping criterion is met
  return The current set of clusters.
```

Clustering usually takes as input a matrix containing numbers indicating the *similarity* or *dissimilarity* of each pair of objects which are to be clustered. Such matrix is called a *proximity matrix.* In RIO, the proximity matrix that is used by the hierarchical clustering contains the *distance measure* for the entity pairs in the ontology signature[4]. Note that the measures used to compute this distance are always symmetrical, therefore the proximity matrix is always a symmetric matrix.

The next question to answer is "what parameters are considered for the computation of the distance between two entities?".

[4]The signature of an ontology is described in Chapter 2, Section 2.3

3.5 Computation of proximity matrix

The *distance* is an important factor in clustering as it captures the similarities of the dataset.

RIO is using the *Jaccard distance* [Han07] which is defined for two sample tests A and B as:

$$J_\delta(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{3.1}$$

Depending on the type of regularities we want to detect, the distance between two entities is measured according to the usage of these entities in a set of axioms or entailments. Thus in RIO, given two entities e_1, e_2 in the signature of an ontology $\mathcal{O}$, their pairwise distance $d(e_1, e_2)$ is computed using the Jaccard distance (1) for which A and B are two sets of axioms defined as follows:

- For the computation of *syntactic regularities*, $A = abs(A_1)$, $B = abs(A_2)$, where A_1 and A_2 are the *asserted axioms* in $\mathcal{O}$ that reference e_1 and e_2 respectively. The *abs* is a replacement function that transforms A_1 and A_2 into abstract forms.

- For the computation of *semantic regularities*, $A = abs(A_1)$, $B = abs(A_2)$, where A_1 and A_2 are the *entailments* in $\mathcal{O}$ that reference e_1 and e_2 respectively. The *abs* is a replacement function that transforms A_1 and A_2 into abstract forms.

The replacement function defined as *abs* above is described in detail in Chapter 4, Section 4.3. At the moment it suffices to know that the purpose of this function is to take as input a set of axioms and return an abstract forms of the axioms by replacing particular entities with general placeholders. This transformation is needed to make two sets of axioms or entailments comparable for the distance computation. In other words, the purpose of this transformation is to capture the similarities between the two entities whose distance is being measured. In Section 4.3 we define different transformation functions and we show how they can affect clustering results. Finally, the proximity matrix includes all the pairwise distances between all the selected entities from the ontology.

3.6 Generalisations

The last step in RIO's workflow is the formulation of generalisations after the completion of clustering. Generalisations are a synthetic view of the clusters in the ontology. A more formal definition was given in Definition 5. A generalisation is an axiom with at least one variable instead of an entity. A generalisation is an abstraction over a set of axioms with similar usage. In RIO, every variable represents a cluster of similar entities, as this was resulted from the cluster analysis. For example, the generalisation:

?AminoAcid SubClassOf AminoAcid

has the ?AminoAcid variable, which represents the cluster that holds all the amino acids.

3.6.1 Generalisation variables

In RIO we borrow the syntax for variables from OPPL. An OPPL variable can be:

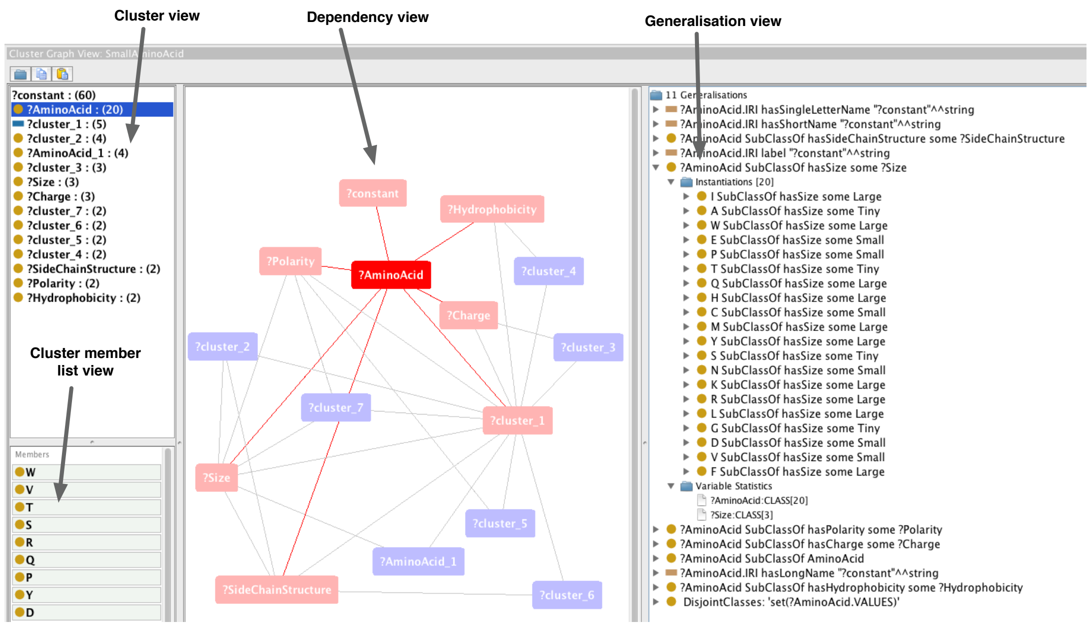
- **Input or Non generated**, i.e., they can replace entities in axioms of the corresponding type (there are OPPL variables for each type of entity in a signature);
- **Generated**, i.e., their value is the result of an expression depending on other variables.

Even though it is not the main focus of the thesis, having an informative label for a variable can facilitate the inspection of generalisations by ontology engineers. In RIO, the label of a variable is selected as follows: Since every variable corresponds to a set of similar entities (cluster), the label of the variable is selected to be the name of the least common subsumer of the entities in the corresponding cluster. If the least common subsumer is the $\top$ (owl:Thing) entity, then a generic name is selected as the label of the cluster variable (e.g. $\mathsf{cluster}_i$, where i denotes the sequence of the cluster). With this approach we try to give a variable name that indicates the type of entities it holds. For example, in the Amino Acid ontology if all Amino Acids are subclasses of the AminoAcid class then the corresponding clusters that include amino acids will be named as ?AminoAcid, ?AminoAcid_1, ?AminoAcid_2 and so on. However, this is not always feasible for all variables as clusters can contain entities that do not share a common subsumer.

3.7 RIO plugin for Protégé 4

The RIO plugin is an Open Source project available online[5]. Its implementation is based on OWLAPI[6]. The generalisations are described using the OPPL vocabulary. A standalone Java tool is used for the regularities' computation, which are then saved in an XML file. The user can load and visualise the regularities in Protégé 4 using the RIO plugin. Figure 3.5 shows one of the main views of the plugin.

The cluster view shows, for every cluster, name and number of entities contained. The Cluster member list view shows the list of entities in a cluster. The generalisation view shows information about the regularities that are captured for every cluster. In the generalisation view, every generalisation can be unfolded to show the axioms that are abstracted (instantiations) and metrics about the variables. The metrics that are shown are the number of instantiations, and the number of entities that are covered by each variable. In principle, the union of the generalisations in the view describes a cluster, but a single generalisation is not necessarily applicable to all members of a cluster.



?constant : (60)
● ?AminoAcid : (20)
▬ ?cluster_1 : (5)
● ?cluster_2 : (4)
● ?AminoAcid_1 : (4)
● ?cluster_3 : (3)
● ?Size : (3)
● ?Charge : (3)
● ?cluster_7 : (2)
● ?cluster_6 : (2)
● ?cluster_5 : (2)
● ?cluster_4 : (2)
● ?SideChainStructure : (2)
● ?Polarity : (2)
● ?Hydrophobicity : (2)

●W
●V
●T
●S
●R
●Q
●P
●Y
●D

11 Generalisations
- ▬ ?AminoAcid.IRI hasSingleLetterName "?constant"^^string
- ▬ ?AminoAcid.IRI hasShortName "?constant"^^string
- ● ?AminoAcid SubClassOf hasSideChainStructure some ?SideChainStructure
- ▬ ?AminoAcid.IRI label "?constant"^^string
- ● ?AminoAcid SubClassOf hasSize some ?Size
  - Instantiations [20]
    - ● I SubClassOf hasSize some Large
    - ● A SubClassOf hasSize some Tiny
    - ● W SubClassOf hasSize some Large
    - ● E SubClassOf hasSize some Small
    - ● P SubClassOf hasSize some Small
    - ● T SubClassOf hasSize some Tiny
    - ● Q SubClassOf hasSize some Large
    - ● H SubClassOf hasSize some Large
    - ● C SubClassOf hasSize some Small
    - ● M SubClassOf hasSize some Large
    - ● Y SubClassOf hasSize some Large
    - ● S SubClassOf hasSize some Tiny
    - ● N SubClassOf hasSize some Small
    - ● K SubClassOf hasSize some Large
    - ● R SubClassOf hasSize some Large
    - ● L SubClassOf hasSize some Large
    - ● G SubClassOf hasSize some Tiny
    - ● D SubClassOf hasSize some Small
    - ● V SubClassOf hasSize some Small
    - ● F SubClassOf hasSize some Large
  - Variable Statistics
    - ?AminoAcid:CLASS[20]
    - ?Size:CLASS[3]
- ● ?AminoAcid SubClassOf hasPolarity some ?Polarity
- ● ?AminoAcid SubClassOf hasCharge some ?Charge
- ● ?AminoAcid SubClassOf AminoAcid
- ▬ ?AminoAcid.IRI hasLongName "?constant"^^string
- ● ?AminoAcid SubClassOf hasHydrophobicity some ?Hydrophobicity
- ● DisjointClasses: 'set(?AminoAcid.VALUES)'

Figure 3.5: RIO main view for Protégé 4.

[5]http://riotool.sourceforge.net
[6]http://owlapi.sourceforge.net

A number of additional views have been developed for viewing stats about the detected regularities.

Software tools that provide visualisations of the regularities in ontologies can be useful as auxiliary services in an ontology editor environment like Protégé 4 and The NeOn Toolkit, as they provide a syntetic view of the large and complex information discovered by tools like RIO. Regularity services can be used to support systematic development, error-fixing and maintenance of an ontology. RIO plugin for Protégé 4 is a prototype to demonstrate the usage of RIO framework and to disseminate it to the community.

3.8 Summary

This chapter started with the description of usage scenarios of a framework like RIO. Then, the challenges on the detection of regularities were presented and finally the overview of RIO and decision options in the architecture were described. RIO can detect two types of regularities; in the asserted axioms of an ontology, named as syntactic regularities, and in the entailments of an ontology, named as semantic regularities. Both types of regularities are detected according to the structural similarities that exist across the axioms. Clustering plays a key role in RIO framework; it is used for detecting groups of entities with similar usage. RIO's workflow is similar to a knowledge discovery (KD) process. The input of RIO is a set of axioms or entailments depending on the type of regularities we want to compute. These axioms are further processed and transformed in order to allow the computation of the proximity matrix consisting of pairwise entity distances. The postprocessing of the results includes the description of the clusters with generalisations; axioms with variables representing a set of similar entities.

The following two chapters present in more detail the methods and algorithms for the computation of syntactic and semantic regularities respectively.

Chapter 4

Computation of Syntactic Regularities

This chapter gives a thorough description of the algorithms for the computation of syntactic regularities, the main functionality of the RIO framework. The detection of syntactic regularities can have multiple uses during the inspection of an ontology as axioms can be grouped according to their similar design. The main goal of the detection of syntactic regularities in RIO is to expose the ontology design style, to abstract over the existing axioms in the ontology and show the general patterns that are used to describe different sets of concepts: it is expected that entities with similar content in an ontology are designed according to the same pattern.

4.1 Syntactic Regularity Detection

As presented in Figure 3.4, four main components are used for the computation of syntactic regularities. These are:

1. Selection of the relevant information from the ontology for the computation of syntactic regularities.
2. Clustering preprocess; proximity matrix calculation.
3. Clustering analysis.
4. Description of clusters with generalisations. These express the syntactic regularities in the ontology.

For the first step, the input that is considered for the computation of syntactic regularities is the set of asserted axioms in an ontology, logical and non-logical (e.g., annotations). The main step in this workflow is the cluster analysis (step 3).

Cluster analysis relies on the notion of distance to quantify how similar (or dissimilar) and, therefore, how close or far apart two entities are in the clustering space. In our workflow, the implementation of Step 2, and, in particular, the distance adopted to compute the proximity matrix, is the most important aspect of the implementation. Even though this step is the preparation of the input for the clustering algorithm, it is more important than the application of clustering, as clustering results rely heavily on the feature measure that is considered [Lan05].

4.2 Distance measure

As described in Chapter 3.5, the calculation of the distance is based on structural similarities between axioms. To compare axioms, we transform them into a more abstract form by applying a *place-holder replacement function* ϕ. We will define the notion of similar axiom structure more formally in the following, and show how this leads to the particular distance we adopted for our cluster analysis.

Using Jaccard's distance (see Equation 3.1) we define the following:

Definition 6 (Distance)**.** *Let $\mathcal{O}$ be an ontology, σ_i and σ_j be two entities from $\mathsf{sig}(\mathcal{O})$, Σ and ϕ a place-holder replacement function. We denote A_i the set $\{\phi_{Ax}(\alpha), \alpha \in \mathcal{O}, \sigma_i \in \mathsf{sig}(\alpha)\}$, i.e: the set of place-holder replacements for the axioms in $\mathcal{O}$ that reference σ_i.*

We define the distance between the two entities, (σ_i,σ_j) $\in \Sigma x \Sigma$ as:

$$d(\sigma_i, \sigma_j) \leftarrow \frac{|\mathsf{A}_i \cup \mathsf{A}_j| - |\mathsf{A}_i \cap \mathsf{A}_j|}{|\mathsf{A}_i \cup \mathsf{A}_j|} \tag{4.1}$$

The distance is computed as an overlap between the two sets of abstracted axioms for each pair of entities in the signature of the ontology. The abstract form of axioms results after applying the place-holder function ϕ in the usage axioms of each entity. From this we can observe that $\forall \mathcal{O}, \phi, e_1, e_2 : 0 \leq d_\phi(e_1, e_2) \leq 1$.

4.2.1 Comparing axioms

Let us consider the famous Pizza Ontology[1]. Its main scope is pizzas and their toppings along with other information such as the country of origin for each pizza, the spiciness of some toppings, or their classification as vegetables, fish, or meat. Among other things, in this ontology we observe that all the topping classes are used as fillers in axioms like these:

(1) *aPizza SubClassOf* hasTopping **some** *aTopping*
(2) *aPizza SubClassOf* hasTopping **only** (*aTopping*
 or *anotherTopping* or ...)

In other words, classes like MozzarellaTopping and TomatoTopping seem similar because they appear as fillers of an existential restriction on the property hasTopping within a sub-class axiom where the left-hand side is a pizza. They also appear as disjuncts in a universal restriction on the same property in another sub-class axiom whose left-hand side is, again, a pizza. Likewise, pizzas in this ontology tend to appear on the left-hand side of sub-class axioms describing their toppings, base, and country of origin.

Therefore, we expect that our cluster analysis should, in the case of the pizza ontology, put together all toppings in a single cluster, pizzas in another, and countries of origin in a third one and so on.

More formally, the distance we introduce quantifies the difference between the usage of two entities in a set of axioms.

4.3 Place-holder replacement function

The place-holder function ϕ enables the comparison between the usage axioms when calculating pairwise distances.

The abstract form of the usage axioms that are generated in this step should not be confused with the final generalisations described in Section 4.7. The generalisations are computed after clustering and provide a description of the clusters while the transformed axioms of this step are only intermediate objects for the computation of the pairwise distances.

Definition 7 (Place-holder replacement). *Let $\mathcal{O}$ be an ontology and let* $\Phi = \{$ *?owlClass, ?owlObjectProperty, ?owlDataProperty, ?owlAnnotationProperty,*

[1]http://www.co-ode.org/ontologies/pizza/

?owlIndividual, ?star $\}$ *be a set of six symbols that do not appear in the signature*[2] *of* $\mathcal{O}$, $\mathsf{sig}(\mathcal{O})$. *A place-holder replacement is a function* $\phi : \mathsf{sig}(\mathcal{O}) \rightarrow \mathsf{sig}(\mathcal{O}) \cup \Phi$ *satisfying the following constraints: Consider* $e \in \mathcal{O}$ *then* $\phi(e) =$

- e *or ?star or ?owlClass, if* e *is a class name;*
- e *or ?star or ?owlObjectProperty, if* e *is an object property name;*
- e *or ?star or ?owlDataProperty, if* e *is a data property name;*
- e *or ?star or ?owlAnnotationProperty, if* e *is an annotation property name;*
- e *or ?star or ?owlIndividual, if* e *is an individual name.*

We define the particular placeholder replacement ϕ^S *as* $\phi^S(e)=$

- *?owlClass, if* e *is a class name (OWLClass type);*
- *?owlObjectProperty, if* e *is an object property name (OWLObjectProperty type);*
- *?owlDataProperty, if* e *is a data property name (OWLDataProperty type);*
- *?owlAnnotationProperty, if* e *is an annotation property name (OWLAnnotationProperty type);*
- *?owlIndividual, if* e *is an individual property name (OWLIndividual type).*

Definition 8 (Place-holder replacement in axioms)**.** *Let* $\mathcal{O}$ *be an ontology,* $\alpha \in \mathcal{O}$ *one of its axioms and* ϕ *a place-holder replacement function. We define* ϕ_{Ax} *as a function that for the input* α *returns a new axiom by applying* ϕ *to all the entities* $e \in \mathsf{sig}(\alpha)$.

ϕ is a way to control the granularity of our distance. It remains to be decided which entities to replace in an axiom.

[2]For signature here we mean the set of class names, data/object/annotation property names, individuals referenced in the axioms of an ontology $\mathcal{O}$.

4.4 Different types of replacement function

Selecting a replacement function ϕ for computing distances is not a straightforward task. That is because the replacement function should transform axioms in a way that captures important semantics. Different heuristics can be adopted for reflecting the most important underlying semantics of the axioms when computing the distance.

In this thesis we propose a tradeoff where we delegate the decision of whether to replace an entity in an axiom to a measure of three different criteria, producing three different replacement functions. These criteria are:

- **Type** of the entity.
- **Popularity** of the entity with respect to the other entities in the same kind of axiom within the ontology.
- Split of entities into variables according to the **structural similarities** of axioms.

Based on these criteria this section presents three different types of replacement function:

1. Naive replacement functions
2. Popularity replacement function
3. Structural replacement function

For each type of replacement function a definition of what is replaced by a general placeholder and an explanation for selecting particular heuristics. In addition, for each case an example is given to demonstrate how the replacement function works when applied to axioms.

4.4.1 Naive replacement approaches

In this approach function ϕ will replace all entities in the signature of an axiom α with a symbol variable. More formally:

Definition 9 (All entity replacement function)**.** *Let $\mathcal{O}$ be an ontology, entity $e \in \mathcal{O}$ and ϕ a replacement function. Consider an axiom α with signature $\Sigma = \mathsf{sig}(\alpha)$, it is a usage axiom of e, for which it is $e \in \Sigma$. An all entity replacement function $\phi_(\alpha)$ with input α will return for every $\sigma_i \in \Sigma$*

- *?star if* $\sigma_i = e$
- $\phi^S(e)$ *otherwise;*

The following example demonstrates the usage of this function when it is used for the computation of a distance between two entities.

Example

Let our $\mathcal{O}$ be the Pizza ontology mentioned above and Margherita, Capricciosa two entities in $\mathcal{O}$. If we want to compute $d(\textsf{Margherita}, \textsf{Capricciosa})$ then ϕ will work as follows: $\forall e \in \mathcal{O}$, $\phi(e)$=

- ?star if $e \in \{\textsf{Margherita}, \textsf{Capricciosa}\}$;
- $\phi^S(e)$ otherwise;

Let us now compute the values of $\phi_{Ax}(\alpha)$ for some of the axioms in $\mathcal{O}$.

α_1 = Margherita *DisjointWith* Cajun
$\phi_{Ax}(\alpha_1)$= ?star *DisjointWith* ?owlClass

α_2 = Capricciosa *DisjointWith* Cajun
$\phi_{Ax}(\alpha_1)$= ?star *DisjointWith* ?owlClass

α_3 = Margherita *SubClassOf* hasTopping **some** TomatoTopping
$\phi_{Ax}(\alpha_3)$= ?star *SubClassOf* ?owlObjectProperty **some** ?owlClass

α_4 = Capricciosa *SubClassOf* hasTopping **some** TomatoTopping
$\phi_{Ax}(\alpha_4)$= ?star *SubClassOf* ?owlObjectProperty **some** ?owlClass

It should be noted that the ?star placeholder represents the entities whose distance is computed. We use this placeholder for keeping track of the position of the entities in the referencing axioms.

Similarly, to compute the distance between $d\{\textsf{TomatoTopping}, \textsf{PizzaBase}\}$ in $\mathcal{O}$, ϕ will be applied as follows: $\forall e \in \mathcal{O}$, $\phi(e)$=

- ?star if $e \in \{\textsf{TomatoTopping}, \textsf{PizzaBase}\}$;

- $\phi^S(e)$ otherwise;

Let us now compute the values of $\phi_{_{Ax}}(\alpha)$ for a pair of axioms in $\mathcal{O}$

α_1 = Margherita *SubClassOf* hasTopping **some** TomatoTopping
$\phi_{Ax}(\alpha_1)$= ?owlClass *SubClassOf* ?owlObjectProperty **some** ?star

α_2 = Pizza *SubClassOf* hasBase **some** PizzaBase
$\phi_{Ax}(\alpha_2)$= $\phi_{Ax}(\alpha_1)$ = ?owlClass *SubClassOf*
?owlObjectProperty **some** ?star

This means that $d(\mathsf{TomatoTopping}, \mathsf{PizzaBase}) < 1$ as it will be from equation 4.1, $|\mathsf{Ax}_\phi(e_1) \cap \mathsf{Ax}_\phi(e_2)| > 0$ and, therefore, $|(\mathsf{Ax}_\phi(e_1) \cup \mathsf{Ax}_\phi(e_2)| - |\mathsf{Ax}_\phi(e_1) \cap \mathsf{Ax}_\phi(e_2)| < |(\mathsf{Ax}_\phi(e_1) \cup \mathsf{Ax}_\phi(e_2)|$.

The consequence would be that distance d_ϕ does not separate as cleanly as possible TomatoTopping (and likewise several sub-classes of PizzaTopping) from PizzaBase, because the selected placeholder function makes their usage axioms to look the same.

Changing the granularity of the place-holder replacement function produces more or less sensitive distance functions. The two extremes are replacing every entity with a place-holder or not replacing any of them. Whilst the former produces a distance that is far too tolerant and puts together entities that seem unrelated, the latter will most likely result in a distance that scores 1 (maximal distance) for most entity pairs.

Property based replacement function

Let $\mathcal{O}$ be an ontology and two entities $\sigma_i, \sigma_j \in \mathsf{sig}(\mathcal{O})$ for which we want to compute $d\{\sigma_i, \sigma_j\}$. For every entity $e \in \mathsf{sig}(\mathcal{O})$ the property based replacement function ϕ will replace the following:

- ?star if $e \in \{\sigma_i, \sigma_j\}$
- e if e is an object property, a data property or an annotation property name;
- $\phi^S(e)$ otherwise;

Example

In this example we will show how the property based replacement function works. For computing the distance $d\{\mathsf{TomatoTopping}, \mathsf{PizzaBase}\}$, ϕ will replace the following:

- ?star if $e \in \{\mathsf{TomatoTopping}, \mathsf{PizzaBase}\}$;
- e if e is a object property name;
- $\phi^S(e)$ otherwise;

Then the values of ϕ_{Ax} for a pair of axioms α will be:

α_1 = Margherita *SubClassOf* hasTopping **some** TomatoTopping
$\phi_{Ax}(\alpha_1)$= ?owlClass *SubClassOf* hasTopping **some** ?star

α_2 = Pizza *SubClassOf* hasBase **some** PizzaBase
$\phi_{Ax}(\alpha_2)$= ?owlClass *SubClassOf* hasBase **some** ?star

The distance will be then $d(\mathsf{TomatoTopping}, \mathsf{PizzaBase}) = 1$. Thus, the entities TomatoTopping, PizzaBase have no similarity.

This replacement function considers all properties as "important" entities in an axiom, thus they are not replaced with a general place-holder as the other entities. The intuition behind this is similar to the detection of isomorphic structures; two graphs are isomorphic when there is an edge-preserving matching of their vertices. Thus, the important part is the connections between the nodes. Considering that axioms can be represented in a form of a graph, a similar approach is adopted for the replacement function.

4.4.2 Popularity based replacement

Definition 10 (Popularity). *Let $\mathcal{O}$ be an ontology, $e \in \mathsf{sig}(\mathcal{O})$ an entity. The place-holder replacement function ${\phi^S}_{Ax}$ for the axioms of $\mathcal{O}$ will extract the* ***structure*** *of each axiom.*

Given an axiom $\alpha \in \mathcal{O}$, let us define the set $\mathsf{Ax}^\alpha = \{\beta \in \mathcal{O}, {\phi^S}_{Ax}(\beta) = {\phi^S}_{Ax}(\alpha)\}$, that is, the set of axioms in $\mathcal{O}$ that have the same ***structure*** *as α.*

We can, finally, define popularity π_{Ax^α} *of an entity* $f \in \mathsf{sig}(\mathcal{O})$ *as*

$$\pi_{\mathsf{Ax}^\alpha}(f) = \frac{|\{\beta \in \mathsf{Ax}^\alpha, f \in \mathsf{sig}(\beta)\}|}{|\mathsf{Ax}^\alpha|}$$

that is, the number of axioms in Ax^α *that reference* f *over the size of* Ax^α *itself.*

We can plug-in popularity as defined above into a place-holder replacement function and therefore in our computation as follows: When computing a distance between two entities, namely e_1 and e_2, for each axiom α where either occurs, the function replaces e_1 or e_2 with ?star and decides whether to replace the other entities with a place-holder depending on their popularity across all the axioms that have the same structure as α.

Definition 11 (Popularity based place-holder replacement)**.** *Let* $\mathcal{O}$ *be an ontology,* $e \in \mathsf{sig}(\mathcal{O})$ *an entity, and* $\alpha \in \mathcal{O}$ *an axiom. Let* Ax^α *and* π_{Ax^α} *be respectively the set of axioms sharing the same structure as* α *and the popularity metric defined in Definition 10. Finally, let* σ *be a function that we call* ***popularity criterion*** *and maps a popularity value into the set* $\{\mathsf{true}, \mathsf{false}\}$.

$\forall f \in \mathsf{sig}(\mathcal{O})$, *we define our function as follows:* $\phi^\alpha_e(f)$

- *?star if* $f = e$;
- f *if* $\sigma(\pi_{\mathsf{Ax}^\alpha}(f)) = \mathsf{true}$;
- $\phi^S(f)$ *otherwise.*

We can now use the popularity based place-holder replacement defined above in our distance (Definition 6). Given two entities e_1 and e_2 according to the formula we need to compute $\mathsf{Ax}_\phi(e_1)$ and $\mathsf{Ax}_\phi(e_2)$. For every axiom α in the ontology $\mathcal{O}$ that references e_1 (resp. e_2), we compute $\phi_{Ax}(\alpha) = \phi^\alpha_{e_1 Ax}(\alpha)$ (resp. $\phi_{Ax}(\alpha) = \phi^\alpha_{e_2 Ax}(\alpha)$). Informally, for each axiom, we compute our replacement function based on the popularity of the entities across the set of axioms sharing the same structure as the axiom we are currently considering.

In the definition above we deliberately parameterised the decision criterion to make our distance framework independent from any particular implementation. In this work, however, we compute a confidence interval $[l, u]$ for the mean value of π_{Ax^α}, (95% confidence). We assume the variance is unknown; therefore in order to compute the area under the distribution function (z), we use the values for the *T distribution*, rather than the *normal* one in the formulas:

$$l = M - z \cdot \frac{\mathsf{sd}}{\sqrt{N}}, \qquad u = M + z \cdot \frac{\mathsf{sd}}{\sqrt{N}}$$

where with sd we denote the standard deviation and with M the mean computed on the set of entities (whose size is N) in the ontology. If the popularity of a given entity is greater than u then we assign true to our σ (see Definition 11), false otherwise.

Example

Once again, let our ontology be the Pizza ontology and let us use as our place-holder replacement function ϕ, the one in Definition 11 (based on popularity).

Let us compute the replacements for the same axioms as in the example in Section 4.4.1. We omit the calculations but the confidence interval for the popularity when applied to such axioms is such that the only entities which will not be replaced are: hasTopping and TomatoTopping, therefore:

$\alpha_1 =$ Margherita *SubClassOf* hasTopping **some** TomatoTopping
$\phi_{Ax}(\alpha_1)=$?owlClass *SubClassOf* hasTopping **some** ?star

$\alpha_2 =$ Pizza *SubClassOf* hasBase **some** PizzaBase
$\phi_{Ax}(\alpha_2)=$?owlClass *SubClassOf* ?owlObjectProperty **some** ?star

The extensive usage of object property hasTopping in this particular kind of axiom is the reason why our place-holder replacement function deems it as important and preserves it in the replacement result.

We observe, however, that deciding replacements based on confidence intervals is strongly dependant on the quality of the sample data. TomatoTopping, for instance, in the example above, is judged *popular* too. The reason is that all pizzas in the ontology have TomatoTopping (and MozzarellaTopping) among their toppings. Conversely, the formula correctly spots that several other entities (Margherita, Pizza, hasBase, ...) are not *relevant* when dealing with axioms presenting a particular structure (?owlClass *SubClassOf* ?owlObjectProperty **some**

?owlClass). We claim that this is preferable w.r.t. making an *a priori* decision, maybe based on users' intuitions, on what should be replaced and when.

4.4.3 Structural replacement function

This approach is based on the search of an optimal split of the entities in corresponding placeholders. We will demonstrate how this transformation policy works using the example ontology in Figure 4.1.

B1 *SubClassOf* B (4.2)
B2 *SubClassOf* B (4.3)
B3 *SubClassOf* B (4.4)
C *SubClassOf* A (4.5)
B *SubClassOf* A (4.6)
C1 *SubClassOf* C (4.7)
C2 *SubClassOf* C (4.8)
C3 *SubClassOf* C (4.9)
C3 *SubClassOf* hasP1 **some** B3 (4.10)
C3 *SubClassOf* hasP1 **only** B3 (4.11)
C4 *SubClassOf* hasP1 **some** B4 (4.12)
C4 *SubClassOf* hasP1 **only** B4 (4.13)
C5 *SubClassOf* hasP1 **some** B1 (4.14)

Figure 4.1: Example ontology

The transformation is done in two steps.

Step 1: The first step is the representation of axioms in abstract forms; This is done by replacing every entity in an axiom with a general variable denoting the type and the position of the entity. Figure 4.2 shows the transformation result for the example ontology.

Step 2: For each one of the general axioms (4.15)-(4.17) we retrieve their instantiations and check if the replacement of a variable with an entity gives better separation of axioms in different groups.

The choice of variable replacements depends on the structural commonalities

$$\text{?class\_2 } \textit{SubClassOf} \text{ ?class\_1} \tag{4.15}$$
$$Instantiations: \ (4.2) - (4.9)$$
$$\text{?class\_2 } \textit{SubClassOf} \text{ ?objectProperty\_1 } \textbf{some} \text{ ?class\_1} \tag{4.16}$$
$$Instantiations: \ (4.10), \ (4.12), \ (4.14)$$
$$\text{?class\_2 } \textit{SubClassOf} \text{ ?objectProperty\_1 } \textbf{only} \text{ ?class\_1} \tag{4.17}$$
$$Instantiations: \ (4.11), \ (4.13)$$

Figure 4.2: Step 1 - Transformation of the axioms in the ontology into abstract forms.

of the axioms. Our criterion is that if there are more than two structural differences between a pair of axioms then the variable should be checked for further replacements. The idea behind this criterion is that we want to find an optimal variable replacement in the axioms that will reflect the differences between the entities in the ontology.

In the example ontology in Figure 4.2, the general axiom (4.15) abstracts the axioms (4.2)-(4.9) of Figure 4.1. Many of these axioms have more than one structural difference ((4.2) and (4.6) or (4.3) and (4.15) etc.). Therefore, further possible replacements should be examined.

The general axiom is the root of the tree. Then, the branches of the tree show all possible values for each variable of the general axiom. An example tree for the generalisation (4.15) is shown in Figure 4.3. The leaf nodes of the tree show the instantiations that result from the replacement of the parent node. Replacements that abstract only a single axiom are discarded. Replacements that separate the values of the other variables into different sets and abstract more than one axiom are kept. For example, in Figure 4.2 all further splits of variable ?class_2 are discarded as they abstract only a single axiom. However, the replacements for ?class_1 are kept as they abstract more than one axiom and separate the values of the other variable (?class_2) into disjoint sets. Therefore, classes A, B and C in the axioms of the form of (4.15) are marked as "relevant" and they are not replaced by a placeholder. The same procedure is followed for the general axioms (4.16) and (4.17). In particular, none of the referenced entities of the general axioms (4.16) and (4.17) are marked as "relevant" because none of the possible replacement abstracts more than one axiom. Thus, all of the referenced entities will be replaced by a placeholder after the application of the replacement function ϕ.

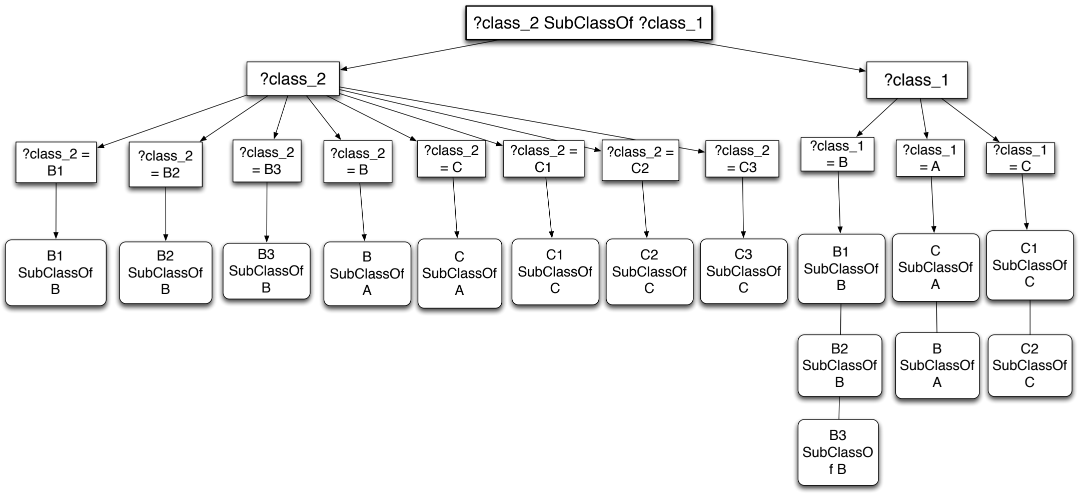


Figure 4.3: Tree showing possible variable replacements.

This replacement policy can be plugged into the placeholder replacement function ϕ. To give an example, the distance $d(\mathsf{B1}, \mathsf{B3}){=}0$ as the application of ϕ on their referencing axioms will give the transformed axioms of Figure 4.4 for both entities.

?star *SubClassOf* B (4.18)
?owlClass *SubClassOf* ?owlObjectProperty **some** ?star (4.19)
?owlClass *SubClassOf* ?owlObjectProperty **only** ?star (4.20)

Figure 4.4: Referencing axioms of entities B1, B3 after the application of the structural replacement function.

4.5 Computation of proximity matrix

Algorithm 2 depicts the computation of the proximity matrix $M_{i,j}$. For the computation of syntactic regularities, Algorithm 2 takes as input an ontology $\mathcal{O}$

and a place-holder replacement function ϕ. It should be noted that one type of replacement function from the ones presented in Section 4.4 is considered in the Algorithm. The output is a symmetric matrix with all pairwise distances between the entities in $\mathsf{sig}(\mathcal{O})$.

Algorithm 2 `ComputeProximityMatrix`$(\phi, \mathcal{O})$

```
Input: A replacement function $\phi$, an ontology $\mathcal{O}$
Output: A proximity Matrix $M_{i,j} = \{m_{i,j}\}$, $0 \leq i,j \leq |\Sigma|$
 1: $\Sigma \leftarrow Sig(\mathcal{O})$
 2: for all $(\sigma_i,\sigma_j) \in \Sigma \text{x} \Sigma$, $0 \leq i,j \leq |\Sigma|$ do
 3:     $\mathsf{Ax}(\sigma_i) \leftarrow$ getAxioms$(\sigma_i, \mathcal{O})$
 4:     $\mathsf{Ax}(\sigma_j) \leftarrow$ getAxioms$(\sigma_j, \mathcal{O})$
 5:     $\mathsf{A}_i \leftarrow \phi(\mathsf{Ax}(\sigma_i))$                          ▷ Transform axioms
 6:     $\mathsf{A}_j \leftarrow \phi(\mathsf{Ax}(\sigma_j))$
 7:     $d(\sigma_i,\sigma_j) \leftarrow \frac{|\mathsf{A}_i \cup \mathsf{A}_j| - |\mathsf{A}_i \cap \mathsf{A}_j|}{|\mathsf{A}_i \cup \mathsf{A}_j|}$     ▷ Calculate distance
 8:     $m_{i,j} = m_{j,i} = d(\sigma_i,\sigma_j)$                 ▷ Build proximity matrix
 9: end for
10: return $M_{i,j}$
```

Algorithm 2 is calling function getAxioms$(\sigma_i, \mathcal{O})$, which returns all axioms from the ontology $\mathcal{O}$ that reference an entity σ_i. In Algorithm 2, Steps 5,6 show the transformation of the usage axioms of every pair of entities (σ_i, σ_j) into abstract forms A_i and A_j respectively. The transformation is achieved with the application of the place-holder replacement function ϕ. Step 7 shows the calculation of the distance and Step 8 shows the construction of each element of the proximity matrix with every distance $d(\sigma_i, \sigma_j)$.

4.6 Agglomerative hierarchical clustering

The clustering algorithm is sketched in Algorithm 3. The input of Algorithm 3 is a proximity matrix $M_{i,j}$ and a stopping criterion P. More details about the stopping criterion P are given later on. Algorithm 3 is calling the sub-routines (1) getElements$(M_{i,j})$, which returns the set of elements from the proximity matrix $M_{i,j}$, which are the entities from the ontology that are considered for clustering; (2) minDistancePair$(M_{i,j})$ which returns the pair of clusters $\langle a, b\rangle$ with the least distance value from the proximity matrix; (3) getClusterFor(a) returns the elements of cluster a; (4) updateProximityWithNewDistances(S) updates the proximity matrix with respect to a set of clusters S.

Algorithm 3 `AgglomerativeHierarchicalClustering(`$M_{i,j}, P$`)`

Input: A proximity matrix $M_{i,j}$, a stopping criterion P
Output: A set of clusters S

```
1: $X_{singletons} \leftarrow$ getElements($M_{i,j}$)
2: $S \leftarrow X_{singletons} = \{\{a\} \mid a \in \Sigma\}$        ▷ Initialise all clusters to be singletons
3: while P is not met do
4:     $\langle a, b\rangle \leftarrow$ minDistancePair($M_{i,j}$)
5:     $S_1 \leftarrow$ getClusterFor($a$)
6:     $S_2 \leftarrow$ getClusterFor($b$)
7:     $S \leftarrow S \cup (S_1 \cup S_2) \setminus \{S_1, S_2\}$
8:     $M_{i,j} \leftarrow$ updateProximityWithNewDistances($S$)
9: end while
10: return S
```

Next we will illustrate how we update the distances in our proximity matrix at every agglomeration and what we use as our stopping criterion.

For the former we use the Lance-Williams formula (also see Section 8.3.3 in [Lan05] - page 524). This formula computes the distance between clusters Q and R, where R is the result of a merger between clusters A and B, as a function of the distances between Q, A, and B. That is

$$p(R,Q) = \alpha_A\ p(A,Q) + \alpha_B\ p(B,Q) + \beta p(A,B) + \gamma|p(A,Q) - p(B,Q)| \quad (4.21)$$

The distance between two sets (clusters) is a function of the distance between their single elements. There are several approaches to compute this, each corresponds to a different value configuration of the coefficients α_A, α_B, β, γ in the general Lance-Williams formula. In the experiments described in the following sections, we used the so-called *centroid* configuration[3]. For this configuration the coefficients are defined as:

$$\alpha_A = \frac{m_A + m_B}{m_A + m_Q} \qquad \alpha_B = \frac{m_B}{m_A + m_B} \qquad \beta = \frac{-m_A m_B}{(m_A + m_B)^2} \qquad \gamma = 0$$

where m_A, m_B, and m_Q are the number of points in clusters A, B, and Q, respectively.

[3]Although vaguely related, not to be confused with a *centroid* in the K-means cluster analysis - see Chapter 8 in [Lan05].

What mainly differs in hierarchical clustering algorithms, is the way the proximity is updated. Different methods such as complete link, single link and group average, will give a different proximity, thus different type of clusters.

As a stopping criterion, our implementation uses a heuristic decision. In particular, the stopping criterion P is selected according to the minimal or maximal differences that can be detected between the elements in the two closest clusters. As defined in Step 7 of Algorithm 2, the value of $d(\sigma_i, \sigma_j)$ is in the interval [0,1].

We use two values as parameters for $P(d(\sigma_i, \sigma_j))$. When $P(0)$(minimal distance differences) is selected (AgglomerateZeros($M_{i,j}$)), the algorithm will stop agglomerations when the distances between all possible pairs of elements for all clusters is equal to 0. When $P(1)$ is selected (AgglomerateAll($M_{i,j}$)), the algorithm will stop agglomerations when the distances between all possible pairs of elements for all clusters is equal to 1 (maximal distance differences).

Definition 12 (Agglomerate decision function). *Let $\mathcal{O}$ be an ontology and d a distance function. We define the function $\mathsf{agg}_d : 2^{\mathsf{sig}(\mathcal{O})} \times 2^{\mathsf{sig}(\mathcal{O})} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ as follows: Given $E = \{e_1, e_2, \ldots, e_n\}$ and $F = \{f_1, f_2, \ldots, f_m\}$ be two clusters, $\mathsf{agg}_d(E, F) =$*

- *If $P(0)$ is selected then*
  - *false, if $\exists 1 \leq i \leq n (\exists 1 \leq j \leq m : d(e_i, f_j) = 0)$;*
  - *true, otherwise.*
- *If $P(1)$ is selected then*
  - *false, if $\exists 1 \leq i \leq n (\exists 1 \leq j \leq m : d(e_i, f_j) = 1)$;*
  - *true, otherwise.*

The algorithm terminates when no pair in the current set of clusters returns true for the Agglomeration decision function agg_d, defined above. In the experiments described in Chapters 7 and 8 $P(1)$ is always selected as a stopping criterion of clustering.

For example, when clustering the Pizza ontology, our implementation returns 17 clusters containing over 110 entities in total; these include:

- A cluster for the toppings that are used in pizzas;

- A cluster for the named pizzas (pizza with a name and a description of their toppings);
- A cluster for the country of origin of the toppings.

As intuitive as these groups may seem, given the average familiarity people have with the pizza domain, this represents a cluster analysis based on the actual usage of the entities in the ontology. In this example clusters seem to follow the taxonomy quite well, however, as we shall see in the experiments chapter, this may not be the case. Performing this kind of analysis can indeed reveal common use between entities that are far apart in the taxonomical hierarchy.

4.7 Generalisations

Once the clusters are available, the axioms that reference entities in the same cluster can be *generalised* and provide a more abstract view on the entire cluster. We can define a generalisation as a simple substitution of an entity with a variable within an axiom. A formal description of generalisations is given in Section 3.6. Here we will give an example of syntactic generalisations from the Pizza ontology.

4.7.1 Example: Generalised Pizzas

Let $\mathcal{O}$ be our Pizza ontology and let $\mathsf{cluster}_1$ be the cluster of all the toppings used in pizzas obtained using our cluster analysis above, and $\mathsf{cluster}_2$ be the cluster of all pizzas. Given $\alpha =$ Margherita *SubClassOf* hasTopping **some** TomatoTopping:

- $g(\alpha, \mathsf{cluster}_1, ?\mathsf{cluster}_1) =$ Margherita *SubClassOf* hasTopping **some** $?\mathsf{cluster}_1$;
- $g(\alpha, \mathsf{cluster}_2, ?\mathsf{cluster}_2) = ?\mathsf{cluster}_2$ *SubClassOf* hasTopping **some** TomatoTopping; or composing the two
- $g(g(\alpha, \mathsf{cluster}_2, ?\mathsf{cluster}_2), \mathsf{cluster}_1, ?\mathsf{cluster}_1) = ?\mathsf{cluster}_2$ *SubClassOf* hasTopping **some** $?\mathsf{cluster}_1$

where $?\mathsf{cluster}_1$ and $?\mathsf{cluster}_2$ are two variables of type class. (In OPPL: $?\mathsf{cluster}_1$:CLASS, $?\mathsf{cluster}_2$:CLASS).

Generalisations provide a synthetic view of all the axioms that contribute to generate a cluster of entities. Each of these axioms can indeed be regarded as an instantiation of a generalisation, as they can be obtained by replacing each variable in g with entities in the signature of the ontology.

4.8 Summary

This Chapter presented the algorithms implemented in the RIO framework for the detection of syntactic regularities. RIO is using cluster analysis for detecting clusters of similar entities. The usage of the clustered entities is expressed in terms of variables representing clusters forming generalisations. The generalisations are the syntactic regularities as every variable in them represents a cluster, i.e. a set of similar entities.

The next Chapter describes the algorithms an various design decisions that are taken in RIO for the detection of regularities in the entailments of an ontology, or semantic regularities.

Chapter 5

Computation of Semantic Regularities

This chapter describes the methods for detecting semantic regularities, i.e., repetitive structures in selected entailments from an ontology. The aim of computing semantic regularities is to gain an overview of possible DL queries - what the ontology talks about, and how to ask for this information. The definitions as well as examples of semantic regularities were given in Chapter 2, Section 3.1. In this Chapter we describe the details of the algorithms and methods that RIO is using for the computation of semantic regularities.

The main difference between the detection of semantic regularities and the detection of syntactic regularities is that for the computation of the semantic regularities a set of entailments needs to be computed from the ontology. This set of entailments is used as an input in the clustering algorithm. Thus, the final regularities are abstractions over the entailments of the ontology and give an overview of the information that can be inferred about similar classes in the ontology; they are not sensitive to the detail of how axioms are expressed, but they capture the meaning of an ontology. As there is, in general, more than one way to entail an axiom from an ontology, using the inferred axioms rather than the asserted axioms means that a regularity can be found across different forms of expression; this regularity would not have been detected if the regularity inspection had stopped at the syntactic level.

To give an example highlighting the difference between regularities detected in the asserted axioms of an ontology and regularities in the entailments of an ontology, consider the axioms of Figure 5.1 taken from the People ontology[1]. Figure 5.1 also shows some of the entailments ((b1)-(b12)) that hold for the axioms (a1)-(a10) of the ontology.

Asserted Axioms:

(a1) van_driver *EquivalentTo* person **and** (drives **some** van)
(a2) lorry_driver *EquivalentTo* person **and** (drives **some** lorry)
(a3) bus_driver *EquivalentTo* person **and** (drives **some** bus)
(a4) haulage_truck_driver *EquivalentTo* person **and** (drives **some** truck) **and** (works_for **some** (part_of **some** haulage_company))
(a5) van_driver *SubClassOf* person
(a6) lorry_driver *SubClassOf* person
(a7) bus_driver *SubClassOf* person
(a8) haulage_truck_driver *SubClassOf* person
(a9) person *SubClassOf* animal
(a10) person *SubClassOf* eats **some** Thing

Entailments:

(b1) van_driver *SubClassOf* animal
(b2) lorry_driver *SubClassOf* animal
(b3) bus_driver *SubClassOf* animal
(b4) haulage_truck_driver *SubClassOf* animal
(b5) van_driver *SubClassOf* eats **some** Thing
(b6) lorry_driver *SubClassOf* eats **some** Thing
(b7) bus_driver *SubClassOf* eats **some** Thing
(b8) haulage_truck_driver *SubClassOf* eats **some** Thing
(b9) van_driver *SubClassOf* drives **some** Thing
(b10) lorry_driver *SubClassOf* drives **some** Thing
(b11) bus_driver *SubClassOf* drives **some** Thing
(b12) haulage_truck_driver *SubClassOf* drives **some** Thing

Figure 5.1: Example asserted axioms taken from the People ontology

[1] `http://owl.man.ac.uk/2006/07/sssw/people.owl`

From the two sets we can derive different types of regularities. Two syntactic regularities that derives from the axioms of Figure 5.1 are shown in Figure 5.2a.

(g1) ?driver *EquivalentTo* person **and** (drives **some** ?vehicle)
  Instantiations: (a1) - (a3)
(g2) ?driver *SubClassOf* person
  Instantiations: (a5) - (a8)

(a) Two syntactic regularities describing the axioms of Figure 5.1.

(g1') ?driver *SubClassOf* animal
  Instantiations: (b1) - (b4)
(g2') ?driver *SubClassOf* eats **some** Thing
  Instantiations: (b6) - (b8)
(g3') ?driver *SubClassOf* drives **some** Thing
  Instantiations: (b9) - (b12)

(b) Three semantic regularities describing the entailments of Figure 5.1.

Figure 5.2: Syntactic and semantic regularities describing the axioms of Figure 5.1.

On the other hand, the regularities that derive from entailments (b1)-(b12) are shown in Figure 5.2b

As it can be seen the semantic regularities are found in the inferred axioms of the ontology, and they abstract on the inferred information. Also information that might not be captured by the syntactic regularities because of the difference in the structure of the asserted axioms can be captured in the semantic regularities. For example, regularity (g1) instantiates axioms (a1)-(a3) but it does not instantiate axiom (a4) even though part of it is that haulage_truck_driver *EquivalentTo* person **and** (drives **some** truck) so it can be covered by the regularity. On the other hand, the semantic regularity (g3') covers instantiations (b9) - (b12), thus it captures the information about the haulage_truck_driver which was excluded in the syntactic regularities. Such additional information can be useful for quality assurance of an ontology; the semantic regularities can give a more complete outline of underlying patterns as these are reflected by the reasoner.

RIO's current implementation is computing a set of entailments using the *Knowledge Explorer* (KE); a toolkit for providing a graph for the entailments of the TBox of an ontology, based on the completion graph generated by the FaCT++ tableaux reasoner [TH06a]. A more detailed description of KE is given in Section 5.1. Details on the algorithms and refinements of methods for the computation of semantic regularities are given in Section 5.2.

5.1 Knowledge Explorer

The knowledge explorer is an extension of the OWLAPI `OWLReasoner`: it is a Java interface that allows client code to explore the completion tree built by a tableaux reasoner. Knowledge explorer has been developed at the University of Manchester by Dimtry Tsarkov and Ignazio Palmisano, the developers of FaCT++ and JFact reasoner respectively. To the best of our knowledge, this interface is currently implemented directly only by the FaCT++ [TH06a] tableaux reasoner. The interface and its documentation are available on the OWLAPI web site[2].

It should be noted that the KE is an interface added in FaCT++ reasoner for simplifying the process of extracting entailments referring to the TBox of the ontology: it provides a simpler and faster interface to retrieve information that a reasoner can provide through the traditional `OWLReasoner` interface. In terms of implementation efficiency, KE provides convenient methods for the computation of entailments occurring between complex classes whose computation with the standard `OWLReasoner` interface is more difficult to implement.

However, KE is not a new implementation of a DL Reasoner. It exploits the implementation of the underlying reasoner (FaCT++ in this context) for answering queries, but does not implement new or alternative reasoning services. Thus, the entailments obtained through KE have the same characteristics of soundness of the implementing reasoner.

Also worth noting is that KE is not tied to FaCT++; it is an extension of the `OWLReasoner` and as such it can be implemented by any reasoner which is accessible through the OWL API. At the time of writing, only JFact[3], the Java

[2]`http://owlapi.sourceforge.net`
[3]`http://jfact.sourceforge.net`

port of FaCT++, has an experimental implementation of the KE; wider diffusion and support will hopefully happen in the near future.

In our approach we want to further exploit the information, which can be exposed by the KE and generalise over the canonical form of the graph. The assumption we make is that the generalisation over the repetitive structures that result from the canonical form of the graph will result to the detection of semantic patterns in the ontology.

The knowledge exploration graph is based on the exploration of a single model[4] that a reasoner builds while it checks the TBox consistency (for details on the computation of TBox consistency the reader should refer to Chapter 2.4). More formally,

Definition 13 (Completion graph). *A* completion graph *is a directed graph* $G = \langle V, E, \mathcal{L} \rangle$*, where* V *is a nonempty set of* nodes*,* $E \subseteq V \times V$ *is a set of* edges*, and* $\mathcal{L}$ *maps every* $v \in V$ *to a set of classes, and every* $e \in E$ *to a set of properties.*

Such a completion graph is produced by a tableaux-based reasoner (e.g., FaCT++) during a satisfiability check. A completion graph corresponding to a satisfiability check for a class A w.r.t. ontology $\mathcal{O}$ (whether or not $\mathcal{O} \models A \sqsubseteq \bot$) has a few features [HKS06]:

1. There is a root node $r \in V$ such that $A \in \mathcal{L}(r)$.
2. For every class $B \in \mathcal{L}(r)$, $O \models A \sqsubseteq B$, where A and B are classes.
3. Every edge in the graph corresponds to a $\exists R.C$ or $\geq nR.C$ class in a label of one of its starting nodes.
4. For every node x and class C, the reasons for $C \in \mathcal{L}(x)$ include:
    - There is a class B: $B \in \mathcal{L}(x)$ and $\mathcal{O} \models B \sqsubseteq C$;
    - There is a node y: $(y, x) \in E$ and $R \in \mathcal{L}(y, x)$ and either $\exists R.C \in \mathcal{L}(y)$ or $\forall R.C \in \mathcal{L}(y)$

Definition 14 (Knowledge Explorer Graph). *A* knowledge explorer graph *is a completion graph whose labelling function* $\mathcal{L}$ *maps every node to a set of* named *classes from Sig($\mathcal{O}$).*

[4]As it has being explained in Section 2.4, a tableaux reasoner creates interpretation models of an ontology.

In essence, the knowledge explorer is a completion graph based on a model[5] for a class A and an ontology $\mathcal{O}$, with class expressions, i.e., non-named classes, removed from the labels of the completion tree nodes. In the future whenever we refer to KE we refer to the exploration of the graph (the letter 'G' is dropped).

An example graph is shown in Figure 5.3, from the people ontology, where the subgraph for the haulage_truck_driver class is shown. The asserted axioms for haulage_truck_driver are shown in Figure 5.1. The root node includes the set of classes $S = \{$animal, person, adult $\}$, which are superclasses of haulage_truck_driver. Some implicit information can be derived from the graph, such as:

$$\mathcal{O} \models \mathsf{haulage\_truck\_driver} \sqsubseteq \exists \mathsf{eats}.\top$$
$$\mathcal{O} \models \mathsf{haulage\_truck\_driver} \sqsubseteq \exists.\mathsf{works\_for}.\exists \mathsf{part\_of}.\mathsf{company}$$
$$\mathcal{O} \models \mathsf{haulage\_truck\_driver} \sqsubseteq \exists \mathsf{drives}.\exists \top$$

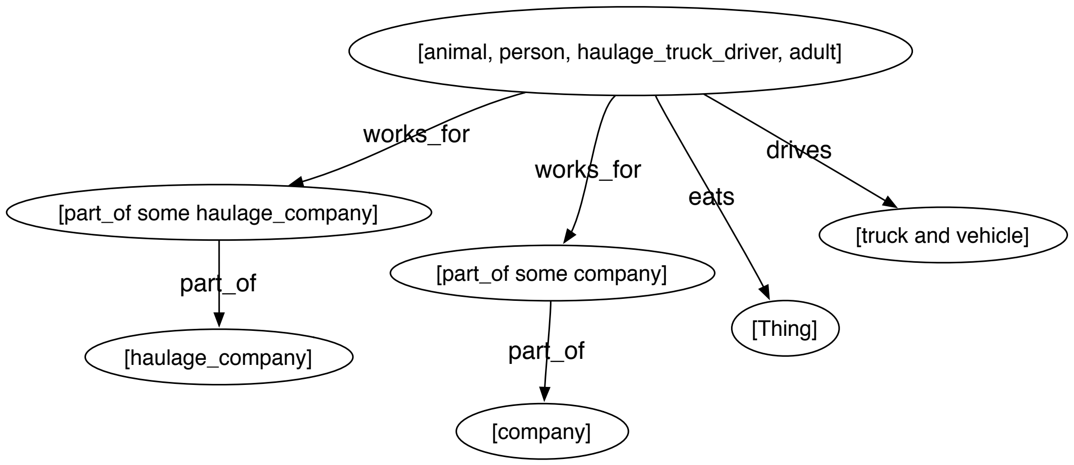


Figure 5.3: An example knowledge exploration subgraph showing possible entailments of the haulage_truck_driver class.

A reasoner implementing KE provides several methods to access the knowledge exploration graph, namely:

- getRootNode(A) returns the root node of a graph for a class A;

[5]An ontology can have infinite interpretation models [BCM+03]. Tableaux reasoners build an interpretation model only for consistent ontologies. For more details we refer the reader to Section 2.4 and [BCM+03].

- getLabel(x) returns a set $\{B : B \in \mathcal{L}(x)\}$
- getProperties(x) returns $\{R : \exists y, (x, y) \in E, R \in \mathcal{L}(x, y)\}$;
- getNeighbours(x, R) returns $\{y : (x, y) \in E, R \in \mathcal{L}(x, y)\}$;

5.1.1 Entailments considered

As it was mentioned the entailments that are extracted from KE are of the form $A \sqsubseteq B$ and $A \equiv B$, where A is always an atomic class and B can be an atomic or complex class, where the grammar for B can be:

$$B \rightarrow \top|A|B \sqcap B|\exists R.B$$

This grammar leads to an infinite instantiations, which is reasonable, due to an infinite number of non-trivial entailments in general. For example, we can have entailments of the form $A \sqsubseteq \exists R.A$, $A \sqsubseteq \exists R.R.A$ and so on. In order to extract a finite set of entailments and to ensure the termination of exploration in KE, we have the following restriction. The models of a concept expression are tree-like. They might have infinite branches which can be unrolled cycles or infinitely long branches. For such cases, the reasoner stops exploring the branch because of a blocking condition [GHM10]. Thus, the KE provides only the part of a branch up to its first repetition. This will cover both infinitely long branches and unrolled cycles. So, if the model looks like $x \rightarrow R \rightarrow y \rightarrow S \rightarrow x' \rightarrow R \rightarrow y' \rightarrow S \rightarrow x'' \ldots$, where $Label(x) = Label(x') = \ldots$ and $Label(y) = Label(y') = \ldots$, the implemented KE will return a concept ($Label(x)$ **and** (**some** R ($Label(y)$ **and** (**some** S $Label(x)$)))), where $Label(x)$ is the label returned by the getLabel(x) function. Such blocking effect prevents from non terminating cycles.

There are also some additional restrictions on the implementation side of KE. At the time of writing, the following information cannot be directly extracted from the KE:

- Expressions containing complex properties. For example, in entailments of the form $\mathcal{O} \models A \sqsubseteq \exists R.C$, if R is a complex property then at the current implementation of KE they are not explored further. Thus, such entailments are skipped.

- In Definition 13 when $\exists R.C \in \mathcal{L}(y)$ or $\forall R.C \in \mathcal{L}(y)$, at the current implementation R is always an object property.

It is worth noting that these limitations are dependent on the FaCT++ implementation, not on the KE interface itself; the authors of the KE are currently working on an update that will remove them.

The reader might argue that the set of extracted entailments is ad hoc for the designed system (KE). For example, in the grammar introduced above it does not include negation or cardinality restrictions. Also entailments on the ABox are not included. However, this type of grammar should suffice for computing entailments in the $\mathcal{EL}$ profile, without any loss of information on the types of entailments with respect to the defined grammar[6]. Considering that many biomedical ontologies like SNOMED-CT, the Foundation Model of Anatomy (FMA), GALEN etc are on this profile, we limit the scope of analysis on this expressivity as it does not constrain us too much. It remains as future work to explore other grammars and compare the significance of results.

5.2 Semantic regularity algorithms

We have described the entailments that are directly exposed by the KE, so now we can proceed with details on the algorithms for the computation of entailments and semantic regularities. The main steps for the computation of semantic regularities are similar to the ones for the computation of syntactic regularities. These are:

1. Extraction of a set of entailments n from the KE.
2. Computation of the proximity matrix of distances between the entities in the $Sig(n)$.
3. Computation of clusters of entities in the $Sig(n)$.
4. Formulation of generalisations that describe clusters of similar entities.

In this section we will not go into details on the clustering algorithm as this has been already described in the previous chapters. The main differences in the algorithms are mainly on the first two steps of the procedure.

[6]$\mathcal{EL}$ profile allows for concept intersections and existential restrictions

Algorithm 4 computes a set of entailments S from the KE. To achieve this, it uses recursive Algorithm 5 to explore all descendant nodes to the RootNode. Algorithm 4 in step 10 calls function checkEntailments (Algorithm 6) to check if the created axioms α, β are entailed by the ontology $\mathcal{O}$. With this method we verify that extracted entailments from the KE are always valid.

Algorithm 4 `ComputeKnowledgeExplorationAxioms`

```
Input: $\mathcal{O}$ an ontology
Output: A set of entailments $S$, such that $\mathcal{O} \models S$
 1: $S \leftarrow \emptyset$
 2: for all $A \in sig(\mathcal{O})$ that $\mathcal{O} \not\models \mathsf{A} \sqsubseteq \bot$ do
 3:     $R \leftarrow$ getRootNode($A$)
 4:     for all $p \in$ getProperties($R$) do
 5:         $F \leftarrow \emptyset$                                          ▷ $p$-fillers
 6:         for all $N \in$ getNeighbours($R, p$) do
 7:             $F \leftarrow F \cup$ getFillers($N$)
 8:         end for
 9:     end for
10:     $S \leftarrow S \cup$ checkEntailments($A, p, F$)
11: end for
12: return $S$
```

Algorithm 5 `getFillers`

```
Input: A node $R$
Output: A set of complex classes Fillers;
 1: Fillers $\leftarrow$ getLabel($R$) $\cup \{\top\}$
 2: for all $p \in$ getProperties($R$) do
 3:     for all $N \in$ getNeighbours($R, p$) do
 4:         for all $C \in$ getFillers($N$) do
 5:             Fillers $\leftarrow$ Fillers $\cup\{\exists p.C\}$
 6:         end for
 7:     end for
 8: end for
 9: return Fillers
```

Algorithm 4 can be used with any consistent OWL-DL ontology as it does not add any constraints on the reasoning process.

The computation of the proximity matrix will be based on the signature of the set of entailments $Sig(S)$. Because we are interested in the computation of semantic regularities, the input of the clustering algorithm should be a set

Algorithm 6 `checkEntailments`

Input: a class A, objectProperty p, a set of classes Fillers;
Output: A set of entailments S, such that $\mathcal{O} \models S$

```
1: $S \leftarrow \emptyset$
2: for all $\mathsf{C} \in$ Fillers do
3:     Axiom $\alpha \leftarrow \mathsf{A} \sqsubseteq \exists \mathsf{p.C} \cup \mathsf{A} \equiv \exists \mathsf{p.C}$
4:     if $\mathcal{O} \models \alpha$ then $S \leftarrow S \cup \{\alpha\}$
5:     end if
6:     Axiom $\beta \leftarrow \mathsf{A} \sqsubseteq \forall \mathsf{p.C} \cup \mathsf{A} \equiv \forall \mathsf{p.C}$
7:     if $\mathcal{O} \models \beta$ then $S \leftarrow S \cup \{\beta\}$
8:     end if
9: end for
10: return $S$
```

of entailments S. Clustering in RIO is based on the computation of pairwise distances of a set of entities. Here, we are interested only in the entities that participate in the entailments, thus the rest of the entities in the $Sig(\mathcal{O})$ are not included.

Algorithm 7 shows the steps for the computation of the proximity matrix $M_{i,j}$. Algorithm 7 is similar to Algorithm 2, only here the calculation of distances is done on the set of entailments. The sub-routine GETENTAILMENTS(σ_i, S) returns the entailments from S that reference the entity σ_i.

Algorithm 7 `ComputeProximityMatrix`(ϕ, S)

Input: A replacement function ϕ, a set of entailments S
Output: A proximity Matrix $M_{i,j} = \{m_{i,j}\}$, $0 \leq i, j \leq |\Sigma|$

```
1: $\Sigma \leftarrow Sig(S)$
2: for all $(\sigma_i, \sigma_j) \in \Sigma x \Sigma$, $0 \leq i, j \leq |\Sigma|$ do
3:     $\mathsf{Ax}(\sigma_i) \leftarrow$ GETENTAILMENTS$(\sigma_i, S)$
4:     $\mathsf{Ax}(\sigma_j) \leftarrow$ GETENTAILMENTS$(\sigma_j, S)$
5:     $\mathsf{A}_i \leftarrow \phi(\mathsf{Ax}(\sigma_i))$                                ▷ Transform entailments
6:     $\mathsf{A}_j \leftarrow \phi(\mathsf{Ax}(\sigma_j))$
7:     $d(\sigma_i, \sigma_j) \leftarrow \frac{|\mathsf{A}_i \cup \mathsf{A}_j| - |\mathsf{A}_i \cap \mathsf{A}_j|}{|\mathsf{A}_i \cup \mathsf{A}_j|}$        ▷ Calculate distance
8:     $m_{i,j} = m_{j,i} = d(\sigma_i, \sigma_j)$                        ▷ Build proximity matrix
9: end for
10: return $M_{i,j}$
```

Finally, function AGGLOMERATIVEHIERARCHICALCLUSTERING$(M_{i,j}P))$, which have been already introduced in Algorithm 3 is called for the computation of the clusters. Algorithm 3 takes as an input the proximity matrix which was computed by Algorithm 7.

5.3 Replacement Function

The replacement functions adopted for the computation of semantic regularities are the same ones defined for the computation of syntactic regularities. The transformation policy is plugged in the replacement function ϕ used in Algorithm 7, Steps 5, 6 for the transformation of the entailments into more abstract forms for the calculation of pairwise distances. Therefore, the following replacement functions are used:

1. Property-based replacement function described in Chapter 4.4.1.
2. Popularity-based replacement function described in Chapter 4.4.2.
3. Structural-based replacement function described in Chapter 4.4.3.

It should be noted that since the set of entailments that are considered for the computation of semantic regularities can contain only object properties, the property-based replacement function will not replace any object properties by placeholders.

The replacement function is applied in the same way for entailments as for the asserted axioms. For example, in Algorithm 7, Steps 5 and 6, the placeholder replacement function ϕ is applied to the referencing entailments $\mathsf{Ax}(\sigma_i)$, $\mathsf{Ax}(\sigma_j)$ $\in S$ of entities (σ_i,σ_j). With the property-based replacement policy, function ϕ replaces entities σ_i, σ_j with ?* placeholder in the entailments. Object properties are not replaced by any placeholder and the remaining entities in the entailments $\mathsf{Ax}(\sigma_i)$, $\mathsf{Ax}(\sigma_j)$ are replaced with a general placeholder denoting their type according to Definition 7.

After the application of the replacement function ϕ, the distance $d(\sigma_i, \sigma_j)$ is calculated as an overlap of the two sets of transformed entailments $\phi(\mathsf{Ax}(\sigma_i))$, $\phi(\mathsf{Ax}(\sigma_j))$. It should be mentioned that this transformation is an intermediate step (Steps 6, 5 in Algorithm 7). The abstract form of entailments that are generated in this step should not be confused with the final generalisations that are produced after the termination of Algorithm 3.

5.4 Entailment Generalisations

Similarly to the computation of syntactic regularities, after the computation of clusters the last step is the formulation of generalisations with respect to the detected clusters. Thus, similarly to the syntactic regularities, a variable in the generalised entailments is representing a corresponding cluster of similar entities. For example, for the people ontology, RIO detected 8 clusters. The results refer to clustering using the popularity replacement function. An example semantic generalisation is shown in Figure 5.4.

Generalisation:
?cluster_2 *SubClassOf* drives **some** (vehicle and ?vehicle)
Instantiations:
van_driver *SubClassOf* drives **some** (van **and** vehicle)
lorry_driver *SubClassOf* drives **some** (lorry **and** vehicle)
haulage_truck_driver *SubClassOf* drives **some** (truck **and** vehicle)
bus_driver *SubClassOf* drives **some** (bus **and** vehicle)

Figure 5.4: An example entailment generalisation and its instantiations from the people ontology.

This generalisation covers 4 entailments from the ontology.

5.5 Summary

This Chapter presented details on the Algorithms for the computation of semantic regularities in ontologies. In particular, it started with the description of the KE which is used for the computation of entailments in an ontology. The computed entailments are used as input of the clustering algorithm, which returns a set of entities that play similar roles in entailments of similar structure. The description of these entailments is given by generalisations.

At this point we have concluded the description of methods for detecting regularities in RIO framework. The next chapters describe methods for evaluating the regularities of RIO and a series of experiments on the application of RIO framework in a variety of real ontologies.

Chapter 6

Evaluation Methods

With the description of RIO framework at hand, we can proceed to the definition of methods for evaluating the validity of the framework and its significance as it is used in a series of experiments. The claims to be demonstrated by experiment in this thesis are:

1. Feasibility of detection of syntactic regularities.
2. Feasibility of detection of semantic regularities.
3. Examination of the cluster validity and assessment of regularity level.
4. Detected regularities can help to reveal the composition style of an ontology.
5. Ontology characterisation in terms of their regular design style.
6. Usefulness of irregularity inspection in an ontology as part of regularity analysis.

In order to verify these claims two types of analysis are performed; *quantitative* and *qualitative* analyses. The aim of the quantitative analysis is to assess validity and reliability of the detected regularities. This is done with a metric scheme presented in this Chapter. The qualitative analysis refers to further analysis and interpretation of the detected regularities in selected ontologies.

This chapter describes methods used for experimental result evaluation. The goal of evaluation is first to validate the clustering results and secondly to verify the validity of the generalisations. This is done with internal criteria such as metrics for assessing the cohesion of clusters and external criteria like projection of generalisations in other independent graphs such as the modular structure.

Statistical evaluation methods are presented in this thesis to investigate the validity of the results. A series of experiments show the usefulness and interpretation of the results such as a systematic inspection of irregularities, performed on SNOMED-CT ontology modules (Chapter 7).

6.1 Motivation

Validation of the results: Since clustering algorithms define clusters that are not known a priori, the final results require some kind of evaluation in most applications [RLR98, HBV01].

In RIO, hierarchical clustering will give a set of clusters of similar entities (according to their usage either in the asserted axioms of the ontology or on a set of entailments). In the worst case scenario, on the last step of agglomeration, every cluster will include a single entity. That means, zero agglomerations occurred during clustering based on the proximity matrix. Thus, a key motivation for cluster validation is that almost every clustering algorithm will find clusters in a data set, even if that data set has no natural cluster structure.

Interpretation of the results: The final product of clustering is the clusters of similar entities as well the description of these clusters with generalisations. The generalisations represent the regularities of the ontology. Assessing the quality of the generalisations, such as size, abstraction level of axioms, cluster coverage etc is important to understand the type of patterns that might exist in the ontology and how these deviate when describing different parts of the ontology.

6.1.1 Cluster validity assessment

One of the most important issues in cluster analysis is the evaluation of clustering results to find the partitioning that best fits the underlying data. This is the main subject of cluster validity [HBV01].

Ideally, the set of clusters should be cohesive and cluster should be well separated from each other. In literature, a first type of experimental validation is the visualisation of clustered data. However, in the case of multidimensional or large scale data this is not the most efficient and systematic way of assessment.

According to literature, there are three approaches to investigate cluster validity [TK06]. The first approach is based on *internal criteria*. That means that the clustering results are assessed in terms of measurements that involve the data set themselves, such as analysis of the proximity matrix, cohesion metrics on the clusters etc. The second approach is based on *external criteria.* This implies that the clustering results are assessed based on a pre-specified structure, which is imposed on a data set and reflects an intuition about the clustering structure of the data set [TK06]. Finally, a *relative criteria* approach compares two structures and measures their relevance.

The cluster validity criteria that are used in this thesis are internal and external.

6.2 Internal criteria

Being able to distinguish whether there is a non-random structure in the data is just one important aspect of cluster validation. This section describes the methods that were used for the quantitative evaluation of the clustering results, known as cluster validity methods. These are mainly metrics which aim to verify that the clusters have not been randomly formed.

6.2.1 Clustering Metrics

As internal criteria we selected metrics that assess the homogeneity of the members of a cluster and the heterogeneity between different clusters. Respectively these metrics measure cluster cohesion and cluster separation. In literature there is a variety of metrics for assessing these aspects [Lan05, HBV01, RH07]. In this thesis the following simple metrics were used:

Definition 15 (Max Internal Distance (MaxID))**.** *Given a cluster C with n elements, we define the Max Internal Distance (MaxID) of C as:*

$$MaxID = \max_{1 \leq i,j \leq n} \{d(x_i, x_j)\}$$

Definition 16 (Min Internal Distance (MinID))**.** *Given a cluster C with n elements, we define the Min Internal Distance (MinID) of C as:*

$$MinID = \min_{1 \leq i,j \leq n, i \neq j} \{d(x_i, x_j)\}$$

Definition 17 (Max External Distance (MaxED))**.** *Given a set of clusters $\{C_i\}, 1 \leq$*

$i \leq n$ we define the Max External Distance (MaxED) as:

$$MaxED = \max_{x \in C_i, y \in C_j, 1 \leq i,j \leq n} \{d(x,y)\}$$

Definition 18 (Min External Distance (MinED))**.** *Given a set of clusters $\{C_i\}, 1 \leq i \leq n$, we define the Min External Distance (MinED) as:*

$$MinED = \min_{x \in C_i, y \in C_j, 1 \leq i,j \leq n, i \neq j} \{d(x,y)\}$$

Definition 19 (Mean Internal Distance (MeanID))**.** *Given a set of K clusters C_K, we define the Mean Internal Distance (MeanID) as:*

$$MeanID = \frac{1}{K} \sum_{m=1}^{K} \frac{\sum_{x_i, x_j, i \neq j}^{|C_m|} d(x_i, x_j)}{|C_m|}$$

where $d(x_i, x_j)$ is the distance between two elements in the cluster C_m and $|C_m|$ is the cardinality of cluster C_m.

Definition 20 (Homogeneity (h))**.** *Given a set of clusters C_n, homogeneity h is defined as:*

$$h = 1 - MeanID$$

and it measures how well formed the clusters are.

Most of these metrics are in the category of overall measures of cohesion and separation, meaning they are weighted sums for assessing the validity of clusters, like the mean internal and mean external distance.

6.2.2 Generalisation quality metrics

In addition, we have defined a set of metrics for assessing the quality of generalisation that describe the cluster. The following metric definitions are used in the results section for assessing the quality of generalisations:

Definition 21 (Cluster Coverage (CC))**.** *Let g be a generalisation, which has a variable v and this variable corresponds to a cluster c with n entities. If v holds m entities from c where $m \leq n$, then the cluster coverage CC_g for the generalisation g is defined as:*

$$CC_g = \frac{m}{n}$$

Definition 22 (Mean Cluster Coverage per generalisation (MCC)). *Given a set of generalisations $G = \{g_1, g_2, \ldots, g_n\}$ for which every generalisation $g_i \in G$ has cluster coverage CC_{g_i}, then MCC is defined as*

$$MCC = \frac{\sum_{i=0}^{n} CC_{g_i}}{|G|}$$

The union of the generalisations describes the cluster, hence a single generalisation might not be necessarily applicable to all the entities in a cluster. Thus, the MCC measures the number of entities in a cluster for which a generalisation is applicable. To demonstrate how it works, consider the example of Figure 6.1.

Cluster: ?Beverage=[Lemonade, Wine, Whiskey, Beer, Coke]
Generalisation:
?Beverage *SubClassOf* contains **some** Alcohol
Instantiations:
Wine *SubClassOf* contains **some** Alcohol
Whiskey *SubClassOf* contains **some** Alcohol
Beer *SubClassOf* contains **some** Alcohol

where ?Beverage=[Wine, Whiskey, Beer]

Figure 6.1: An example cluster and generalisation describing beverages

In this example, cluster ?Beverage has 5 members in total. However, the example generalisation is applicable only for 3 of these classes. Thus, the cluster coverage by this generalisation is 60%.

Definition 23 (Mean Instantiations per Generalisation (MI)). *Given a set of generalisations G which cover A axioms (instantiations), we define*

$$MI = \frac{|A|}{|G|}$$

MI is a measure that shows the level of abstraction for each generalisation.

The MI metric is an important one used in all of the experiments of this thesis, as an estimation of the abstraction impact of a generalisation over an ontology.

It essentially measures the number of axioms or entailments from the ontology, that is abstracted by a generalisation. In the above example, the generalisation ?Beverage *SubClassOf* contains **some** Alcohol has 3 instantiations. In cases that the MI metric is high, it indicates that a dominant regularity exists that is captured by a few generalisations. More details on the uniformity of an ontology are given in the following section.

6.2.3 Uniformity

The main question about detection of regularities is which strategy captures regularities, if they exist, in the most efficient way. Then a second question that arises is what is considered as an efficient way. An ontology can be one of the following in terms of regularities:

- It can be irregular
- It can be regular with many different forms of regularities
- It can be regular with a few different forms of regularities

According to the RIO framework, we can characterise an ontology as irregular if all the generalisations in the ontology cover only single axioms. This is an extreme case, which is unlikely to happen for medium to large size ontologies. The reason is that the axioms are constructed following a syntax, thus they are expected to have some syntactic regularities.

We define *uniformity* in regularities as the *degree of diversity of regularities* in an ontology. The intuition behind uniformity is to define a characteristic that allows the assessment of the detected regularities. According to the previous states, an ontology can be regular, but have different forms of regularity; thus it has low uniformity. Uniformity can give an intuition of the compositional complexity of an ontology. It shows that different design decisions were taken for describing different portions of the ontology. For example, a general pattern that has been chosen to describe a set of entities in the ontology, can have some deviations when it is applied in a different set of entities. On the other hand, an ontology can be regular with a high level of uniformity, meaning that the same form of regularity appears in most axioms of the ontology. On a second level, this reveals a low compositional complexity of the ontology.

According to the RIO framework, we can observe the following properties in order to assess the uniformity of an ontology:

- Number of generalisations
- Number of instantiations per generalisation
- Cluster coverage by generalisations
- Degree of homogeneity of clusters

An indicator of a regular and homogeneous ontology is the number of generalisations covering a high number of instantiations. According to these two criteria we define uniformity as the average of the MI and MCC metric, thus:

$$uniformity = \frac{MI + MCC}{2}$$

The higher the number of instantiations covered by a generalisation, the more regular the ontology is; a side effect is that the number of such generalisations must be small. As a consequence, the cluster coverage by per generalisation will be close to 1. On the other hand, the existence of a high number of generalisations of similar structures gives the indication of a regular but not very uniform ontology. For example consider the following two generalisations:

?Pizza *SubClassOf* hasTopping **some** ?Topping_1
?Pizza *SubClassOf* hasTopping **some** ?Topping_2

These indicate the existence of two clusters of toppings; ?Topping_1 and ?Topping_2 which results in two generalisations even though they have similar structure. On the other hand, a single generalisation describing this type of regularity would have been:

?Pizza *SubClassOf* hasTopping **some** ?Topping

The high number of generalisations of similar structure having only a few differences in the variables, indicates the existence of a regularity with many deviations. There can be two explanations for the deviations. The first one is that the distance approach which is selected does not capture similarities between entities in the most efficient way but the detected regularities is an approximation. The second one is that entities that result in the same cluster share common axioms expressed by a generalisation. However, this does not exclude the existence of axioms of different structure using entities of the same cluster. For example, in the above generalisations ?Topping_1 and ?Topping_2 even though they are toppings, they can hold different types, such as meaty toppings and vegetarian toppings respectively. This distinction is reflected on the referencing axioms of these entities and it is the reason that separates them in different clusters.

6.3 External criteria

Cluster validity measures are intended to help us measure the goodness of the clusters that we have obtained. Indeed, they typically give us a single number as a measure of that goodness. However, we are then faced with the problem of interpreting the significance of this number, a task that may be even more difficult [Lan05].

External criteria of evaluation include methods which are more heuristic and tailored to the nature of the data. In our case, we are dealing with OWL ontologies, which are known to be complex logical artifacts. OWL ontologies consist of structured data. Different structural analysis of an ontology can be applied. In this work, we are using the modular structure of an ontology to check if it complies with the detected syntactic and semantic regularities. This structure has been also described in Chapter 2.5 as the Atomic Decomposition. Even though there is no one to one correspondence between the detected regularities and the modular structure exposed by the Atomic Decomposition, there are similarities between them. For example, regular axioms have also an impact on the structure of the Atomic Decomposition. Details about these similarities are discussed later on.

It should be noted that in the external evaluation we focus more on the precision of detecting regularities which are expected to be found rather than recall.

6.3.1 Syntactic regularities external evaluation

This section includes other experimental methods for the validation of the results. In literature [Lan05], external evaluation evaluates clustering results using data that was not used for clustering, such as known class labels and external benchmarks. Such external benchmarks are predefined classifications on the data. In practice, it is quite difficult to have a predefined classification of the entities in the ontology, as most of the times, the purpose of clustering is knowledge acquisition.

Reference to ontology documentation or developer

On the qualitative analysis of the ontologies, dominant patterns described in the documentation of the ontologies were compared with the detected syntactic regularities. The documentation of the ontologies in this case worked as a gold standard for the verification of the results. However, this is just a limited method of verification as in very few cases we could refer to the documentation of the ontologies, which described a pattern. The results of such analysis are described in Chapters 7 and 8.

Verification with OPPL scripts

Ontology documentation in many cases is quite abstract and will give only an intuition of how the ontology is structured. For evaluating the difference between the detected regularities and the expected pattern, sometimes this documentation does not suffice. That might be because there might be a difference between the intended design as it is described in the documentation and the actual design of the ontology. For example, there might be cases in which the pattern prescribed in the ontology technical guide is not respected in a large number of cases in the actual ontology. In Chapter 2 we introduced the usage of OPPL as a language for manipulating ontologies and ingesting patterns in them.

In Chapter 7 we show a qualitative analysis of the syntactic regularities in three modules from SNOMED-CT. These modules are expected to have axioms that instantiate patterns described in the technical guide of the ontology [sno11]. For a more systematic verification of the results, we initially express these patterns with OPPL to count their instantiations in the ontology and then we compare these results with the regularities detected from RIO.

Projection of generalisations in the atomic decomposition

In Chapter 2, an introduction to Modularity and Atomic Decomposition was presented. The Atomic Decomposition shows the modular structure of an ontology. It is a dependency graph between atoms and each atom is a maximal set of axioms which co-occur in ontology modules.

As an evaluation method, the AD is independent from clustering results. Of course, the input in both structures is the same ontology but the methods for partitioning this input are different. There is prior evidence that AD can expose the semantics of the ontology by showing the dependency between atoms [VPS12, VPS11]. An atom with dependencies cannot exist on its own in a module; the dependent atoms will also included in the module. That indicates that AD shows how axioms have logical dependencies to other axioms, with respect to a modularisation algorithm; both have been proved to be sound and complete in terms of what can be inferred from the ontology, therefore the structure highlighted by the AD is semantically sound and not an artifact of the syntactic structure of the ontology.

By showing a high correlation between the AD structure and the generalisations found by RIO, we are highlighting the fact that RIO generalisations are capturing semantically sound repetitions, not just syntactic regularities which are incidental to the ontology intended semantics, thus confirming the reliability of the detected patterns.

The AD of the ontology $\mathcal{O}$ of Figure 6.2a is shown in Figure 6.3. This ontology consists only of subclass axioms. The axiom in A7 is built upon A2, which means that a module that contains axioms of A7 will also contain the axiom of A2. This structural observation can be also explained from a cognitive aspect; the description of Breaststroke_swimming is built upon the description of Swimming, which is included in A2.

To evaluate the quality of regularities, we project the final generalisations in AD to assess the compression of this structure; the intuition behind this is that atoms with regular structure will be merged after the projection of regularities in the AD. This shows that the detected generalisations are not random but they correspond to repetitive structures also reflected in the AD structure. In other words, verifying this repetition with AD is an indication that the detected generalisations are meaningful. Thus we define AD compression as:

$$ADCompression = \frac{|GADatoms|}{|ADatoms|}$$

where *ADatoms* are the atoms of the Atomic Decomposition, and *GADatoms* are the atoms whose axioms have been replaced with generalisations.

The syntactic regularities of the example ontology $\mathcal{O}$ as they can be detected by RIO using the popularity-based placeholder function are presented in Figure 6.2b. The projection of these generalisations in the AD of the ontology will cause compression of the structure, shown in Figure 6.4. The projection of the

Cycling *SubClassOf* Sport
Swimming *SubClassOf* Sport
  Freestyle_swimming *SubClassOf* Swimming
  Breaststroke_swimming *SubClassOf* Swimming
Sightseeing *SubClassOf* Hobby
Painting *SubClassOf* Hobby
  Oil_painting *SubClassOf* Painting
  Watercolor_painting *SubClassOf* Painting

(a) Example ontology.

Generalisation:
?Hobby *SubClassOf* Hobby
Instantiations:
Sightseeing *SubClassOf* Hobby
Painting *SubClassOf* Hobby
{?Hobby:CLASS=[Sightseeing, Painting]}

Generalisation:
?Painting *SubClassOf* ?Hobby
Instantiations:
Oil painting *SubClassOf* Painting
Watercolor painting *SubClassOf* Painting
{?Painting:CLASS=[Oil painting, Watercolor painting], ?Hobby:CLASS=[Painting]}

Generalisation:
?Sport *SubClassOf* Sport
Instantiations:
Swimming *SubClassOf* Sport
Cycling *SubClassOf* Sport
{?Sport:CLASS=[Swimming, Cycling]}

Generalisation:
?Swimming *SubClassOf* ?Sport
Instantiations:
Freestyle swimming *SubClassOf* Swimming
Breaststroke swimming *SubClassOf* Swimming
{?Swimming:CLASS=[Freestyle swimming, Breaststroke swimming], ?Sport:CLASS=[Swimming]}

(b) Four regularities and corresponding instantiations of the example ontology detected by RIO using popularity replacement function.

Figure 6.2: Example ontology and its syntactic regularities detected by RIO

generalisations caused a 50% compression of the AD. It should be noted that the resulting structure has no longer the same dependency structure as the initial AD. For example, atoms A1 and A3 do not have any dependents in the AD but they get merged with A2 and A4 respectively. However, there are structural similarities on the initial AD and these are captured by the generalisations.

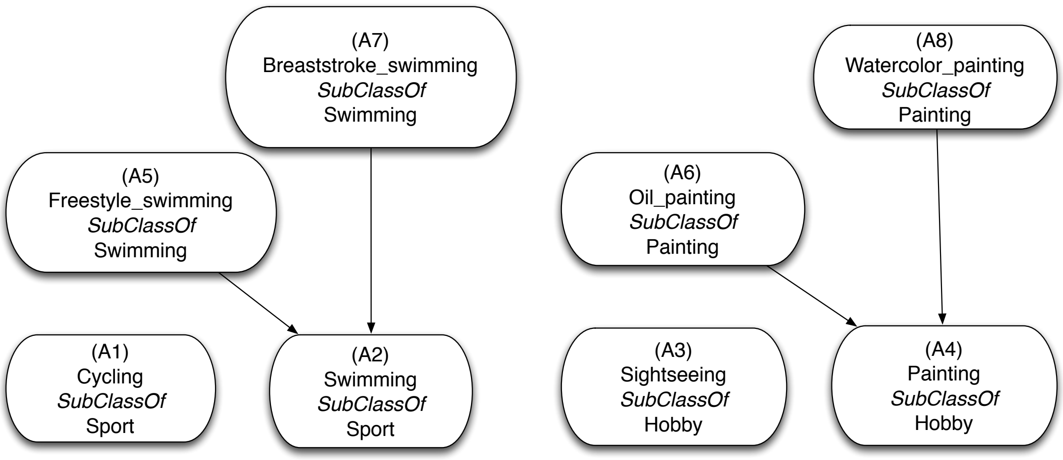


Figure 6.3: AD of the example ontology

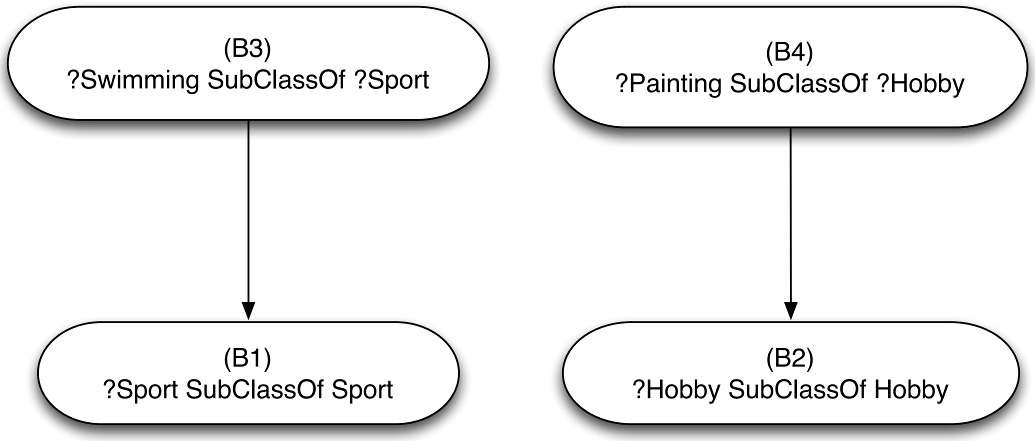


Figure 6.4: Compressed AD of the example ontology $\mathcal{O}$. In this example, the generalisation are detected using RIO clustering with popularity replacement function.

In addition, different types of isomorphic structures can be detected in Figure 6.3. Someone could argue that since the ontology has only subclass axioms the main syntactic regularity is ?x *SubClassOf* ?z, with ?x and ?z holding the corresponding entities from the ontology. Thus, everything should be merged in one atom. The regularities expressed by the generalisations are also verified by the AD compression. However, the ones detected by RIO with the popularity placeholder replacement take into account the popularity of various entities in the ontology, therefore not everything is getting merged.

6.3.2 Semantic regularities evaluation

For the evaluation of the semantic regularities the same internal and external criteria are used. For the external criteria the following are taken into account:

AD considers only the asserted axioms of an ontology. However, modules that are extracted according to the AD ensure the preservation of all entailments over the signature of the module [VPS12], therefore we do not have to worry about losing entailments by using AD modules. In our evaluation approach, the set of entailments that result from parsing the KE, are used for creating an AD of entailments. Then, the detected generalisations are projected to this AD and the compression of the structure is measured in the same way as with the syntactic regularities.

6.4 Pattern induction

As it has been mentioned in the previous sections, a pattern in this thesis is considered as a design template for describing a set of entities in the signature of an ontology. The term pattern is distinguished from the term Ontology Design Pattern (ODP) as not necessarily all patterns in an ontology are best practices to a problem; they are however design templates adopted by the developers of an ontology. As a consequence, the instantiation of a pattern will give rise to repetitive structures in an ontology.

A single regularity resembles a repetitive structure in the axioms of an ontology. It can resemble a pattern or not. The cluster analysis that RIO performs in an ontology, will have as a result a set of clusters of similar entities whose description is given by a set of generalisations. These generalisations might resemble one or more patterns. In practice, it is easier to distinguish these patterns in ontologies with few axioms, as there are fewer generalisations per cluster. However, as it is shown in the experiments of Chapter 6, big ontologies result in many generalisations. Even though their number is much smaller than the initial number of axioms, it is difficult to distinguish clear patterns that completely describe a set of entities. The set of generalisations describing a cluster is close to be considered as a pattern, however, there are two limitations:

- A dominant regularity might have caused the merge of many entities in a cluster. Apart from this regularity, there can be many deviations on the remaining generalisation that describe the cluster.
- The variables of a single generalisation, which correspond to a cluster, more often do not cover all the members of that cluster but a subset of those. Thus, it needs further filtering of these generalisations and separations to distinguish patterns that completely describe a set of entities.

An approach for filtering the detected regularities, so to distinguish patterns, would be to try combinations of regularities according to the axioms they instantiate. For example, lets assume we have following cluster and its description shown in Figure 6.5.

Generalisations (3) and (4) do not cover all entities in the ?Beverage cluster. Thus, two patterns that can be distinguished in this cluster are shown in Figure 6.6.

Pattern 1:
?Beverage *SubClassOf* Beverage
?Beverage *SubClassOf* hasColor **some** ?Color
?Beverage *SubClassOf* contains **some** Alcohol

where {?Beverage:CLASS=[Beer, Wine], ?Color=[Yellow, Red]}

Pattern 2:
?Beverage *SubClassOf* Beverage
?Beverage *SubClassOf* hasColor **some** ?Color
?Beverage *SubClassOf* contains **some** Carbonated_water

where {?Beverage:CLASS=[Lemonade, Coke], ?Color=[White, Black]}

Figure 6.6: Two patterns describing alcoholic and non-alchoholic beverages from Figure 6.5.

As it is shown, these two patterns share the first two generalisations and deviate in the third generalisation. Thus, trying all combinations of generalisations

Cluster: ?Beverage:CLASS{Beer, Wine, Lemonade, Coke}

Description:
(1) ?Beverage *SubClassOf* Beverage
Instantiations:
Beer *SubClassOf* Beverage
Wine *SubClassOf* Beverage
Lemonade *SubClassOf* Beverage
Coke *SubClassOf* Beverage

(2) ?Beverage *SubClassOf* hasColor **some** ?Color
Instantiations:
Beer *SubClassOf* hasColor **some** Yellow
Wine *SubClassOf* hasColor **some** Red
Lemonade *SubClassOf* hasColor **some** White
Coke *SubClassOf* hasColor **some** Black

(3) ?Beverage *SubClassOf* contains **some** Alcohol
Instantiations:
Beer *SubClassOf* contains **some** Alcohol
Whiskey *SubClassOf* contains **some** Alcohol

(4) ?Beverage *SubClassOf* contains **some** Carbonated_water
Instantiations:
Lemonade *SubClassOf* contains **some** Carbonated_water
Coke *SubClassOf* contains **some** Carbonated_water

Figure 6.5: An example cluster and its description.

in a cluster can lead to the distinction of patterns. However, such procedure can be computationally impractical especially for big ontologies, as they have quite many generalisations.

The Atomic Decomposition can facilitate such process. In particular, the projection of the resulting generalisations on the AD will cause the compression of this structure only in places where repetitive structures exist; atoms with same number and type of generalisations will be merged. Thus, if a generalisation is projected in n atoms and m of these are getting merged, where $m < n$, it indicates a pattern shared across the signature of the m atoms. Therefore, parsing the compressed structure will distinguish the different types of patterns that exist in the ontology, and highlight their deviations. In the experiments presented in

Chapters 7, 8 we explore patterns that occur across merged atoms. We name such patterns as *atom patterns*.

6.5 Summary

This chapter described the various methods for evaluating the results of the experiments described in Chapter 7. These are distinguished in two categories; internal and external criteria. The internal criteria include metrics for the clusters derive from the proximity matrix to give an intuition of the distances between the data. How close they are, and how well separated are the clusters from each other. Internal criteria also includes metrics for assessing the quality of the final generalisations; how many they are, how many axioms they instantiate etc. The external criteria on the other hand include comparison with manual detection of pattern either via documentation and OPPL scripts or by metrics and comparison with the modular structure of an ontology. The following Chapter shows these criteria in action, through a series of experiments using a variety of real ontologies.

Chapter 7

Experiments on Syntactic Regularities

This Chapter and the next one describe the experiments demonstrating the usage of the RIO framework meets its aims. So far we have described the methods used in RIO for the detection of syntactic and semantic regularities. The detection of syntactic regularities creates an abstraction over the asserted form of an ontology, while the detection of semantic regularities creates an abstraction over the entailments of an ontology.

7.1 Experiments Setup

The setup of the experiments for both syntactic and semantic regularities is the same. This Chapter presents the results on the syntactic regularities while Chapter 8 presents the results on the semantic regularities. The results are presented and discussed in a similar way for this and the following Chapter.

Two types of analysis are used in the experiments on the detection of regularities[1]. These are:

1. **Quantitative analysis** on a larger collection of ontologies from the BioPortal repository [2]; a repository consisting of bioinformatics and biomedical ontologies. The aim of this analysis is to examine whether the results can be generalised over a larger set of ontologies. In this analysis, an evaluation of the framework is performed according to the evaluation metrics defined

[1]both syntactic and semantic.

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

in Chapter 6. In addition, a characterisation of the ontologies is done with respect to their patterns and their uniformity[3].

2. **Qualitative analysis** by using a small set of ontologies for which we have additional information about their construction. These are ontologies for which the documentation is accessible or the ontology developers can be consulted. The aim of this analysis is to examine in depth the results of RIO and perform additional tasks such as inspection of irregularities, quality assurance of the ontology, and so on, and through this to demonstrate the potential of the RIO framework for the detection of dominant patterns in the ontology as well as analysis of their deviations.

7.1.1 Clustering Tasks

The main clustering algorithm used by RIO, presented in Chapter 3, is Agglomerative Hierarchical Clustering (AHC).

Three replacement methods were introduced in Sections 4.4.1, 4.4.2 and 4.4.3 named as *property based replacement*, *popularity based replacement* and *structural replacement function* respectively.

These methods influence the distance granularity during the computation of the proximity matrix, thus affecting the shape of the final clustering results. Details of these methods are described in Chapter 4.

Based on these variations the following three clustering tasks are performed for each experiment:

1. Clustering using the popularity replacement function.

2. Clustering using the structural replacement function.

3. Clustering using the property relevance function.

7.1.2 Quantitative Analysis: BioPortal Repository

The following setup is the same for the quantitative analysis in both syntactic and semantic regularities.

[3]The uniformity has been defined in Chapter 6.2.3.

Curation Procedure

The BioPortal [4] ontologies used in this experiment were last updated on December 2011 using the BioPortal RESTful Service API. 208 OWL ontologies were downloaded for the experiment; non-compliant ontologies, and ontologies that could not be parsed or had missing imports were excluded from the experiment.

By non-compliant ontologies, we mean ontologies outside OWL DL and OWL 2 DL profiles, usually because of the wrong use of some OWL constructs. A common violation was the use of non-simple properties in axioms that only allow for simple properties, and the lack of declarations for classes or properties. The first violation caused an ontology to be excluded, as it is hard to decide how to correct the ontology; the second class of violations is easier to fix, therefore we proceeded to fix the affected ontologies instead of dropping them from the experimental set.

Other Parameters of the Setup

Timeout consideration Due to practical considerations, a total timeout of 45 minutes of CPU time was set for the above clustering tasks for each ontology in the quantititative analysis. In both qualitative experiments (for syntactic and semantic regularities) the same timeout was defined for the generation of the results from the BioPortal corpus. In both cases an acceptable number of ontologies were processed varying in size and DL expressivity, thus the timeout was not further increased for processing more ontologies.

Evaluation parameters The following evaluation steps are not evaluation of ontologies using RIO but rather evaluation of RIO using ontologies. Thus, for each task the metrics described in Chapter 6 were computed for assessing:

1. The quality and validity of the clusters
2. The quality and validity of the generalisations
3. The clustering method that gives the best results
4. The characterisation of ontologies with respect to their regularities

Presentation of Results For the analysis of the syntactic and semantic regularities of the BioPortal ontologies, the same type of graphs and additional

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

statistical tests are used for the observation of the results and for the verification of the hypothesis that derive from the observation. These are:

1. Table summarising the total average values for each evaluation regularity metric. Such table gives an initial numerical intuition on the shape of the regularities that were generated for each clustering task.

2. Graphs representing the values of each evaluation metric, grouped by clustering task for all ontologies. The aim of such graph is to give more details to help the observation and the formulation of hypothesis about the results.

3. Boxplotss [MTL78], which are more abstract than the graphs representing the values for each metric for every ontology. Boxplots are used to gain an intuition of the distribution of values for each metric.A box plot is a graph depicting five descriptive stats:

   (a) **Min:** It represents the smallest value.

   (b) **Lower Quartile (Q1):** 25% of the data has a value smaller than this value in the plot.

   (c) **Median (Q2):** This is the middle value of the data set. 50% of the data has a value higher than this value.

   (d) **Upper Quartile (Q3):** 25% of the data has a value higher than this value.

   (e) **Max:** The greatest value.

4. T-Test for verifying hypothesis formulated with the observation of the previous graph. In particular t-Test is used for verifying the significance of difference between the results of different clustering tasks.

5. Pearson's correlation metric [Wei03] for verifying correlation between different evaluation metrics. Thus, the picture of regularities gained from the observation of the graphs is verified with t-Test for comparison between the results of different clustering tasks and Pearson's correlation for correlating results on the same clustering task.

This Chapter starts with the presentation of the main experimental setup and then for each experiment the presentation and discussion of results follows.

7.2 Experiment 1: Syntactic Regularity Detection in BioPortal Ontologies

This section describes the quantitative analysis of the detection of syntactic regularities in the BioPortal repository. Initially we will describe the ontologies used for the experiment and then the presentation and discussion of the results.

7.2.1 Results

RIO completed all three syntactic clustering tasks within the allotted timeout for a set of 93 ontologies. This section presents the results of these tasks and a discussion with additional analysis of the results follows.

Table 7.1 shows the total mean values for selected metrics for the processed BioPortal ontologies. The results of this table indicate that RIO detected syntactic regularities in all of the ontologies of the corpus with all methods.

Metrics	**Popularity**	**Structural**	**Property Relevance**
Generalised Axioms (%)	52%	51%	33%
Instantiations per Generalisation	5.5	5.49	5.07
# Clusters	67.66	59.88	8.065
# Entities per Cluster	5.88	8.09	31.287
Cluster Coverage (%)	54%	52%	40%
Homogeneity	0.60	0.53	0.77
AD Compression (%)	54%	55%	19%

Table 7.1: Total mean values for selected regularity metrics for the 93 processed ontologies from BioPortal repository.

It can be observed from Table 7.1 that the results for the popularity replacement function and structural replacement function are closer than the ones of the property based relevance replacement function. For example, the total mean percentage of the generalised axioms for both popularity and structural replacement method is almost the same (52% and 51%). However, it is markedly lower for the property replacement method (33%). Thus, a first hypothesis derived from the summary of total averages in Table 7.1, is that when structural and popularity replacements are used, the clustering algorithm will detect more regularities in the axioms of an ontology.

A more detailed view of these metrics for all three replacement methods are presented in Figures 7.1–7.5 for each ontology. In all Figures, the results are sorted in ascending order according to the axiom number of the ontologies. In particular, the processed ontologies were categorised as small, medium and large according to their axiom numbers. As it is also indicated in the x-axis of the figures, small ontologies were in the spectrum of 30-300 axioms, medium ontologies were in the range of 300-1000 axioms and finally big ontologies had more than 1000 axioms. Figure 7.1 shows the percentage of generalised axioms; meaning the percentage of axioms from the ontology instantiating a regularity. Figure 7.2 shows the Mean Instantiations per Generalisations for each ontology, or in other words the abstraction impact per generalisation. It should be noted that a higher scale on the y-axis is used for ontologies 47-93 of Figure 7.2. Figure 7.3 shows the mean cluster coverage per generalisation. This metric shows how many entities from a cluster are covered in average from the corresponding variable that is referenced in a generalisation. Figure 7.4 shows the homogeneity of the corpus. Finally, Figure 7.5 shows the Atomic Decomposition (AD) compression for each ontology from the corpus. As it has been described in Chapter 6, Section 6.3.1, the AD compression metric is an external criterion for assessing the quality of generalisations with respect to the compression they cause when projected in the AD.

7.2.2 Discussion

Comparison of replacement methods

Table 7.2 shows the p value for the Student's t-Test revealing the significant difference on metric values for different replacement methods[5]. Each column on the table represents a potential hypothesis for a pair of replacement methods; that one of the two methods is always greater than the other. The *null hypothesis* H_0 is that no difference can be drawn between the two methods of the pair. When $p < 0.05$, then H_0 is rejected.

From Figures 7.1-7.5 and Table 7.2 the following hypotheses can be verified:

- H_1**:** The percentage of generalised axioms with the structural replacement function is always greater than the one resulted with the popularity and

[5]The t-test is between "paired" values, and one tail type, meaning that one of the methods is always lower or greater than the second.

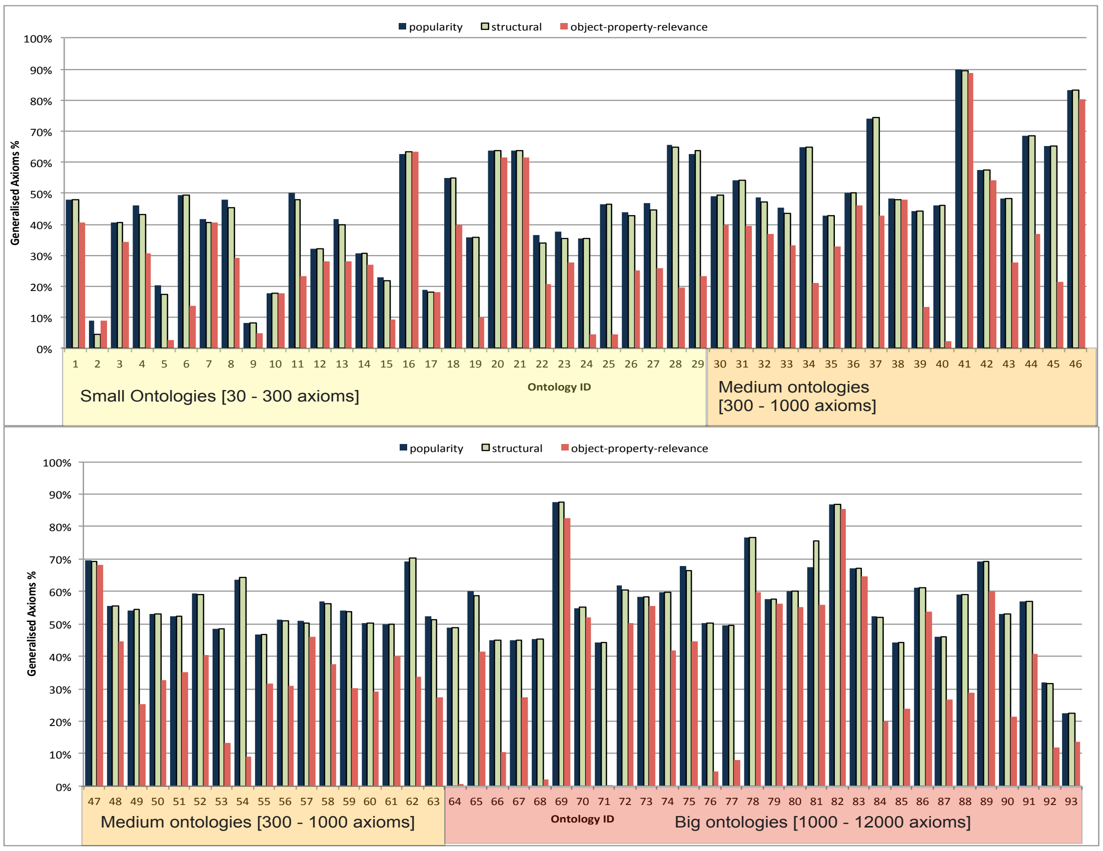


Figure 7.1: Percentage of abstracted axioms in BioPortal ontologies

Metric	Popularity - Property	Structural-Property	Popularity - Structural
Generalised Axioms	$1.067 \cdot 10^{-21}$	$5.365 \cdot 10^{-21}$	$1.035 \cdot 10^{-2}$
Mean Instantiations per generalisation	0.077	0.226	0.00021
Cluster Coverage	0.00498	$1.3 \cdot 10^{-10}$	0.004989
AD Compression	$5.54 \cdot 10^{-22}$	$3.024 \cdot 10^{-27}$	0.034
Homogeneity	$2.8 \cdot 10^{-11}$	$1.56 \cdot 10^{-20}$	$8.38 \cdot 10^{-10}$

Table 7.2: T-Test results for selected metrics. The table shows the p value for checking the significance of difference in regularity metrics between different replacement methods that were used in clustering.

property replacement function. This is depicted in Figure 7.1.

- H_2: Clustering with the structural replacement function resulted in higher instantiations per generalisation than clustering with popularity (p=0.00021). It is almost valid that popularity replacement function will give generalisations with greater abstraction impact (mean instantiations per generalisation) than the property replacement function, while no conclusion can

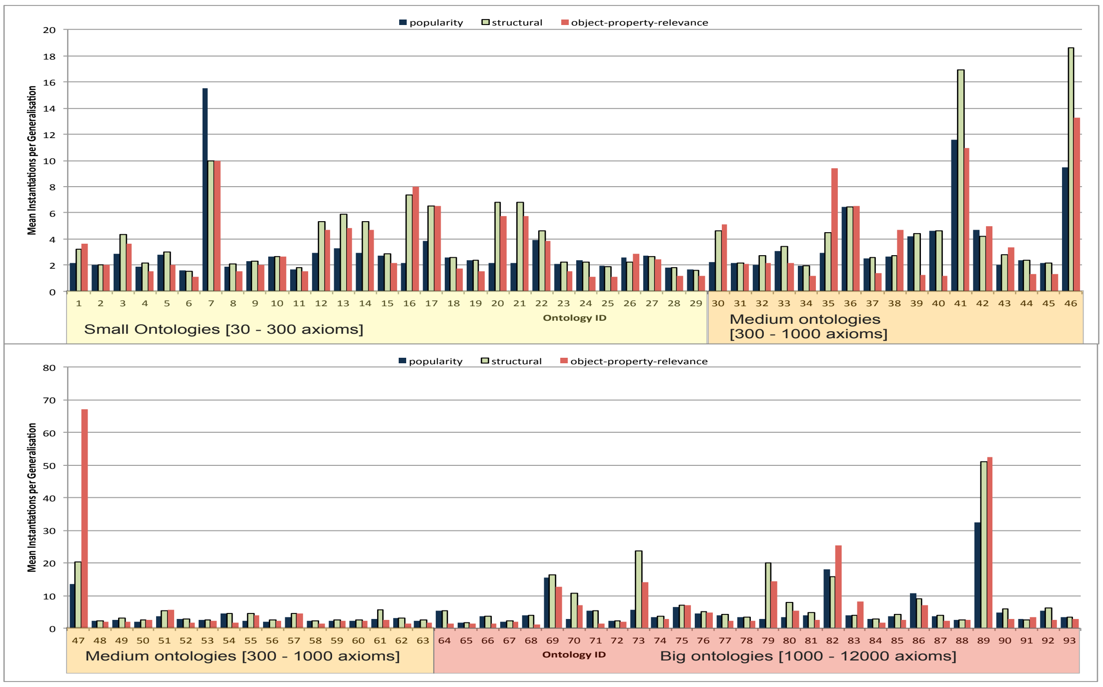


Figure 7.2: Mean Instantiation per Generalisation of BioPortal Ontologies.

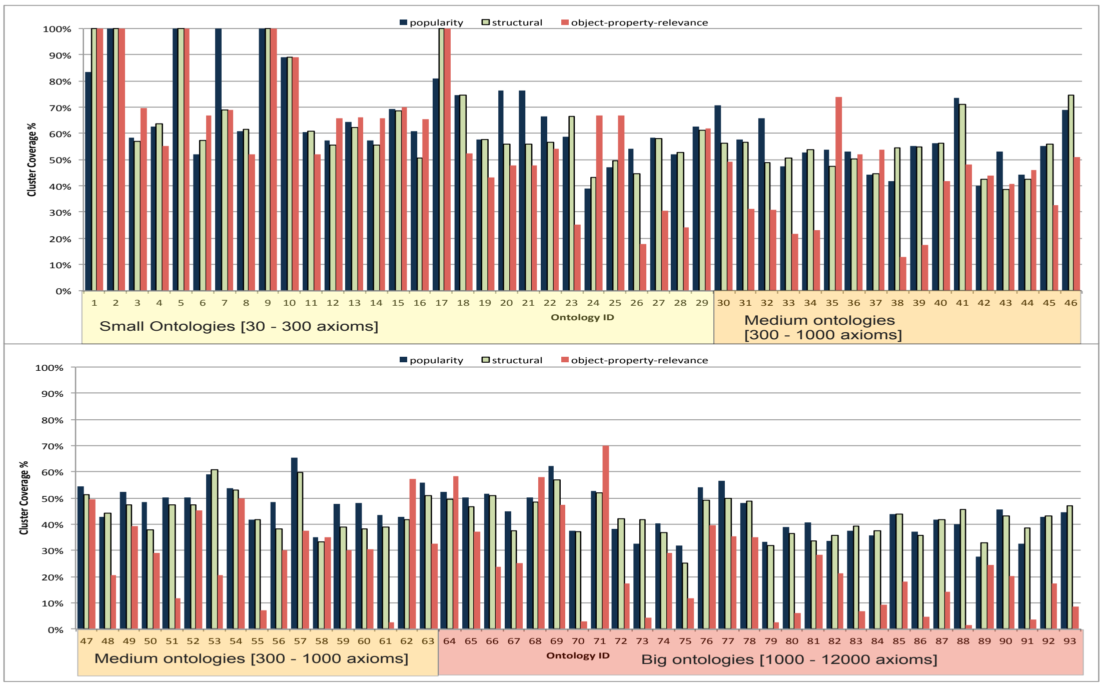


Figure 7.3: Mean Cluster Coverage of BioPortal Ontologies.

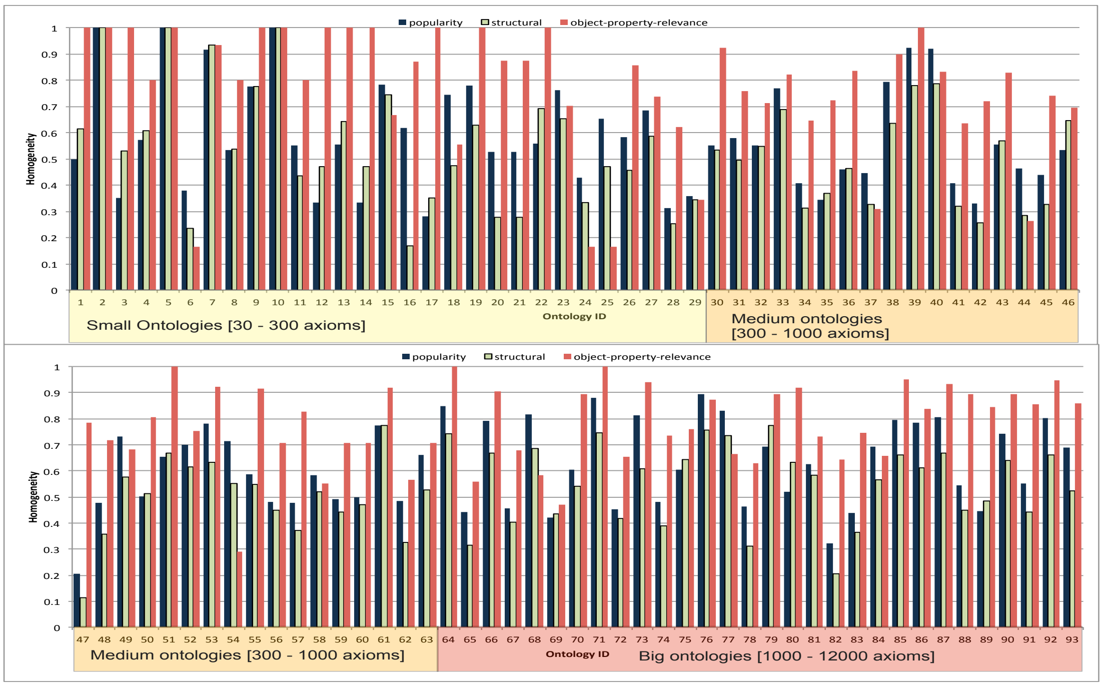


Figure 7.4: Homogeneity of BioPortal Ontologies.

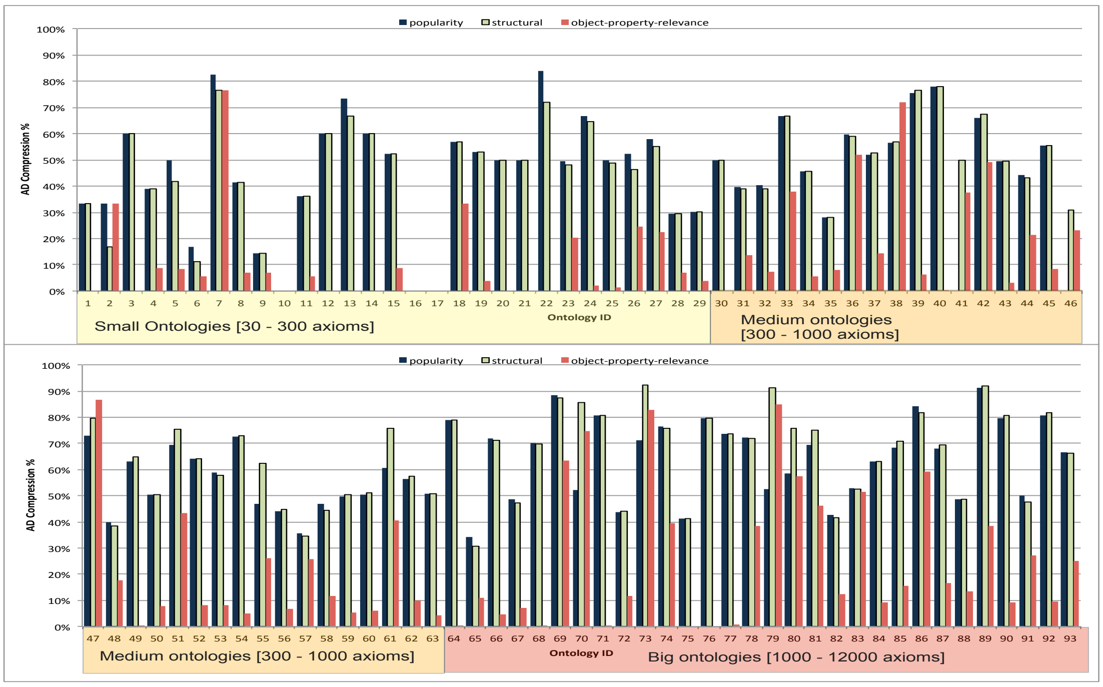


Figure 7.5: AD Compression of BioPortal Ontologies.

be drawn on this metric between the structural and property replacement functions.

- H_3**:** Clustering with popularity returns regularities with higher cluster coverage than clustering with the property replacement function and clustering with the structural replacement function. Similarly the cluster coverage for clustering with the structural replacement function is higher than clustering with the property replacement function.

- H_4**:** Clustering with the structural replacement function results in generalisation that can cause the highest AD compression compared to the other two. Similarly, clustering with the popularity replacement function results in generalisations causing higher compression in AD graph than the clustering with the property function.

- H_5**:** Clustering with property replacement function returns clusters with the highest homogeneity. In addition, clustering with popularity returns more homogeneous clusters than clustering with the structural replacement function.

Finally a descriptive summary of the dataset on selected metrics is shown in Figures 7.6-7.10. The boxplots of these Figures can give a better intuition on the distribution of values on regularity for each metric and type of replacement function.

In general, popularity and structural methods had for the majority of ontologies comparable results (Figures 7.6-7.10). However, clustering using the property relevance method results in the inclusion of fewer axioms under a regularity, as shown in Figure 7.1 and depicted in H_1. Also, although the difference is not verified by the t-test, for many of the ontologies the number of instantiations per generalisations is lower than in the other two methods. In general, the property-based replacement function will lead to a more sensitive distance, meaning more pairs of entities will be dissimilar. Thus, fewer but more homogeneous clusters are formed (as hypothesis H_5 is valid) and as a result fewer regularities are detected in the ontology (H_1). These regularities are more fine-grained than the ones detected with the other two methods; the mean instantiations per generalisation and the mean cluster coverage per generalisation for the property replacement function are lower (shown in Figures 7.2 and7.3 respectively).

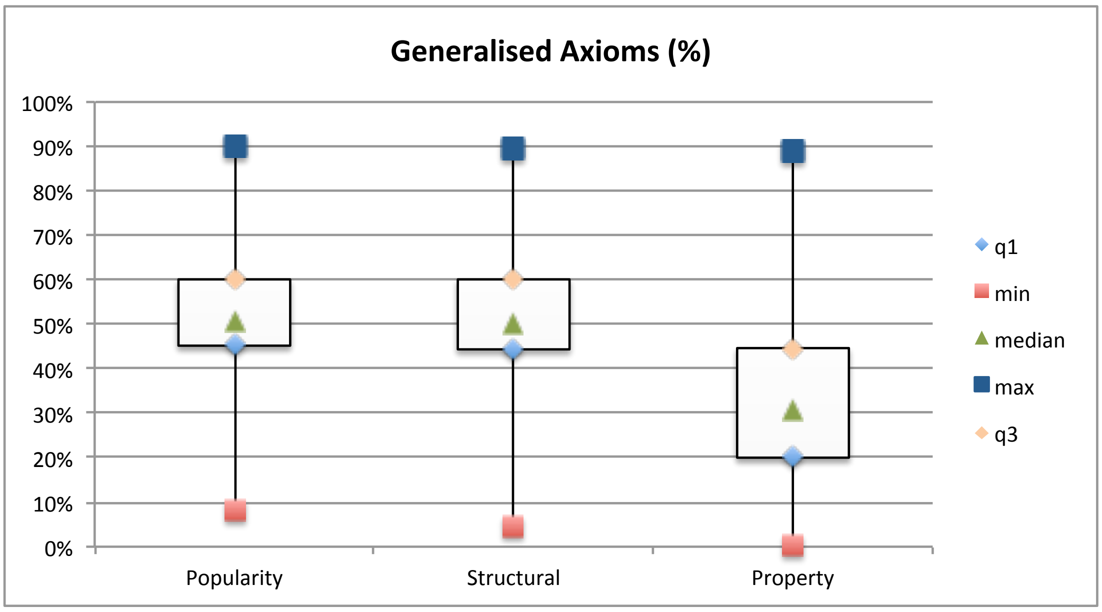


Figure 7.6: Boxplot on generalised axioms

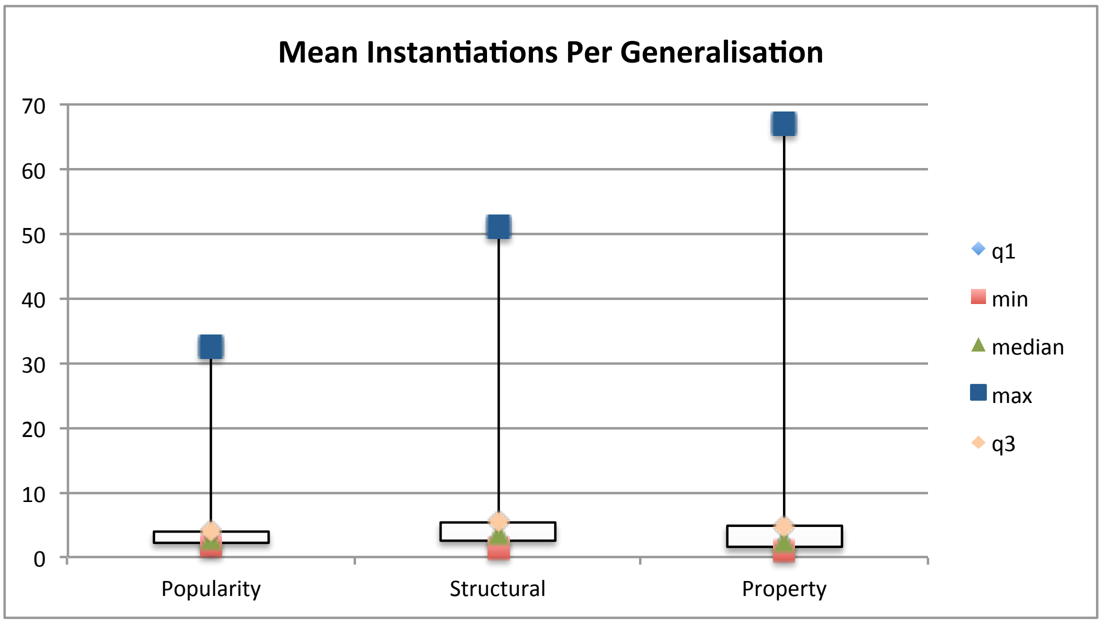


Figure 7.7: BoxPlot on Mean Instantiation per Generalisation of the BioPortal Ontologies.

Projection of generalisations in the AD

In a nutshell, the higher percentage compression of AD is an indication of good quality of generalisations; meaning that the repetitive structures expressed by

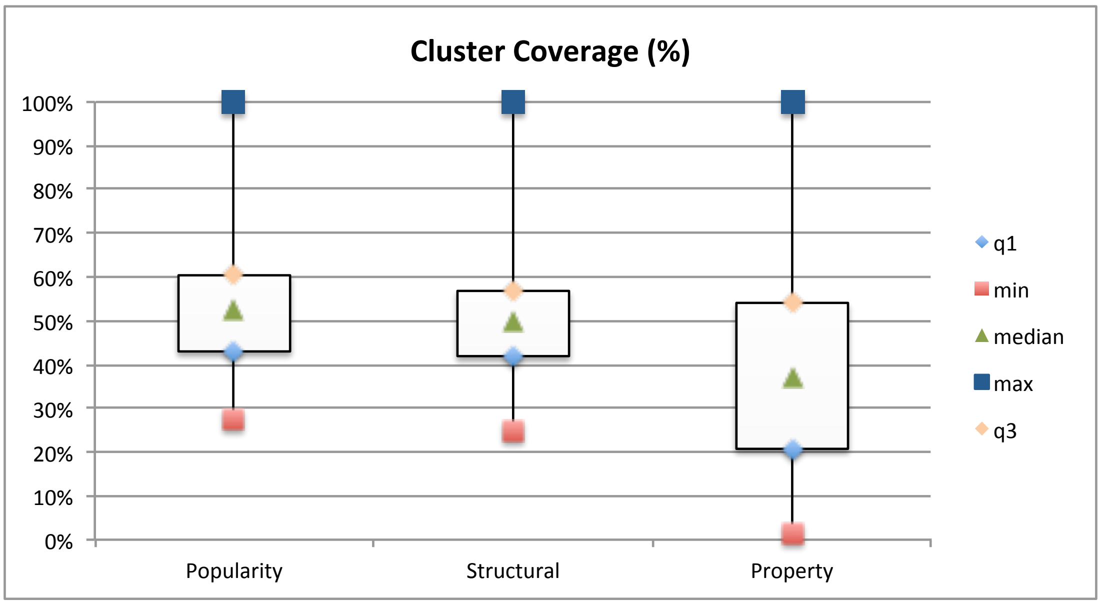


Figure 7.8: Boxplot on Cluster Coverage of BioPortal Ontologies.

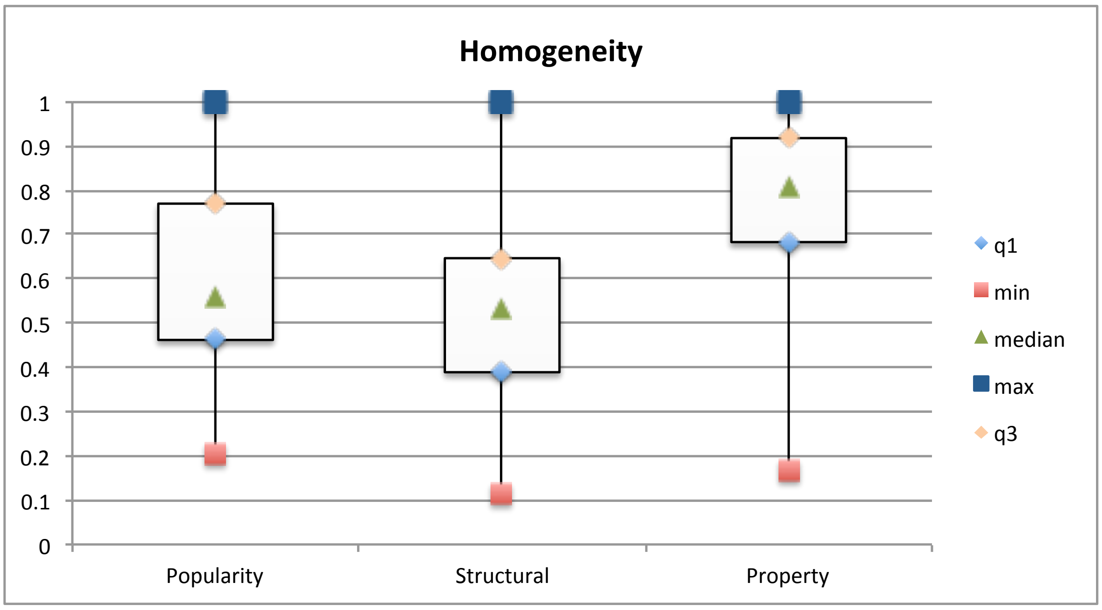


Figure 7.9: Boxplot on the homogeneity of BioPortal ontologies.

the generalisations are also reflected in the AD dependency graph. Of the 93 ontologies used in the experiment, 33 of them exhibit less than 50% AD compression, and 9 had this metric below 30%. These results refer to clustering with the popularity replacement function. Figure 7.10 shows that the distribution of

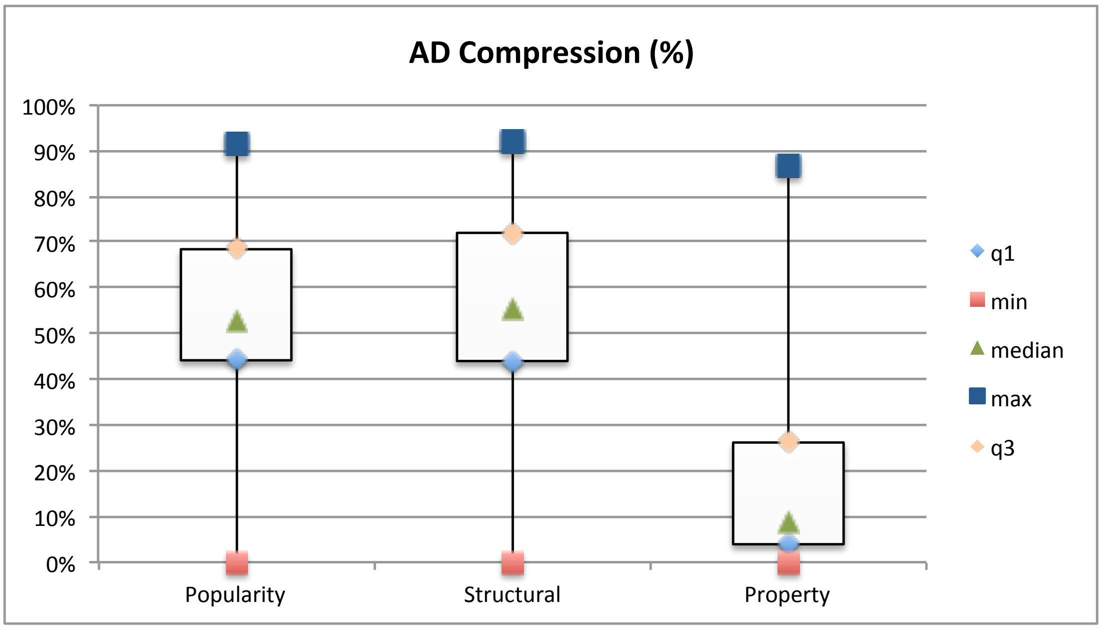


Figure 7.10: Boxplot on AD Compression of the BioPortal Ontologies.

the results for the popularity replacement function is of similar shape as the one for the structural replacement function. However, the results of the AD compression for the popularity replacement function are lower than the other two methods (H_4). In the results of the popularity method, five of the ontologies had 0% compression but these had on average more than 50 axioms per atom. On the structural replacement, the AD compression results are very similar with the ones of the popularity measure; from these only 3 of the ontologies have 0%. On the property relevance method, 19 had AD compression 0% and 72 were above 30%. The intuition behind the definition of the AD compression metric was that meaningful generalisations will cause higher compression on the AD, since the same repetitive structures also exist in the AD. However, such claim needs further investigation, as there can be a case of a regular ontology that does not cause any merge on the atoms of the AD.

Regularity and Uniformity of Ontologies

All 93 ontologies appear to have some form of regularity. The big ontologies in the corpus (more than 1000 axioms) had an average of 43% of their axioms instantiating a regularity. For the big ontologies, a single generalisation was instantiated on average by 5.8 axioms in the ontology with minimal 1.7 and maximal 10.6 axioms when using the popularity replacement function. Small ontologies instantiated 2.9 axioms per generalisation and medium ontologies 3.6 axioms per generalisation. For the structural replacement function, big ontologies instantiated 8.7 axioms per generalisation, medium ontologies instantiated 4.7 axioms per generalisation and small 3.7 axioms per generalisation.

Ontology 89 (cognitive-atlas) is an example of a uniform ontology; it has on average 32.3 axioms per generalisation, with mean cluster coverage 28% and AD compression 91%. Even though there is not a clear correlation between the number of instantiations per generalisation and the AD compression (verified using Pearson's correlation), there are cases like ontologies 89, 47 and 73 for which a high number of instantiations per generalisation was followed by a high AD compression.

Ontologies with a high number of instantiations followed by a high cluster coverage indicate a regular and uniform ontology. Figure 7.11 shows the mean cluster coverage as well as the mean instantiations per generalisation for clustering with the structural replacement function. We selected the structural replacement function since in the majority of evaluation metrics seems to give better results than the other two methods. Figure 7.11 reveals uniform ontologies as they have both values higher than the others. Based on this combination of metrics, Table 7.3 shows detailed regularity metrics on the 7 most uniform ontologies in the corpus.

Ontology/ Axioms	**#Clusters**	**Entities per Cluster**	**Cluster Coverage(%)**	**Instantiations per Generalisation**	**Homogeneity**
7 / 74	1	15	69%	10	0.93
41 / 530	7	6.9	71%	16.93	0.32
46 / 579	10	5.8	74%	18.6	0.65
47 / 589	4	31	51%	20.35	0.11
69 / 1 270	13	8.54	57%	16.32	0.43
73 / 1 624	18	34.6	42%	23.68	0.61
82 / 3 146	30	12.3	36%	15.7	0.21

Table 7.3: Regular and uniform ontologies from the BioPortal corpus. The results for the structural method are shown.

Uniform Ontologies

Table 7.3 contains uniform ontologies from all categories of size (small, medium, big). Ontology 73 (tick-gross-anatomy) has 1624 axioms and has the highest number of instantiations per generalisation from all the 93 processed ontologies.

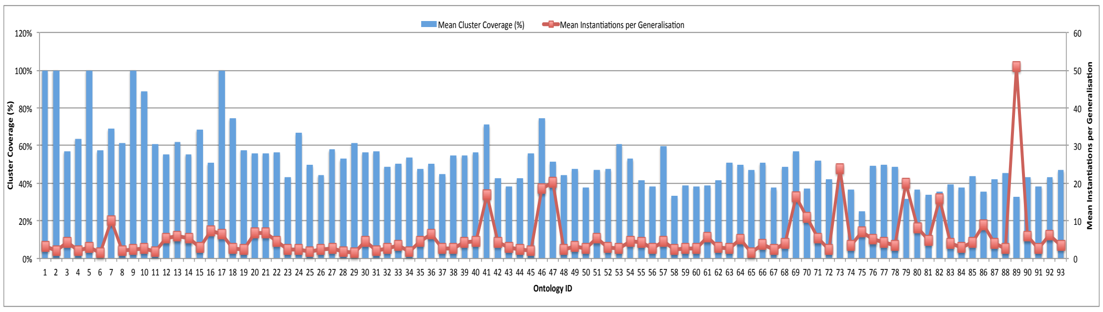


Figure 7.11: Uniformity indication. Cluster coverage and mean instantiations per generalisations for the structural method

Indeed, this ontology has a highly regular and homogeneous pattern referring to existential restrictions using the part_of relationship. Four generalisations were found with such a structure, abstracting 582 axioms from the ontology. Such a regularity is shown in Figure 7.12 and it covers 425 axioms in the ontology. Other detected regularities in this ontology were referring to atomic subsumptions. Such a generalisation (2) is shown in Figure 7.12 instantiating 96 axioms in the ontology.

(1) ?cluster_1 *SubClassOf* part_of **some** ?cluster_2
Total Instantiations: (425)
Example Instantiation:
adult hemocyte *SubClassOf* part_of **some** adult circulatory system
where {?cluster_1:CLASS=[adult hemocyte],
?cluster_2:CLASS=[adult circulatory system]}

(2) ?cluster_1 *SubClassOf* ?anatomical_structure
Total Instantiations: (96)
Example Instantiation:
adult ovary funicular cell *SubClassOf* cell
where {?cluster_1:CLASS=[adult ovary funicular cell],
?anatomical_structure:CLASS=[cell]}

Figure 7.12: Example regularities with many instantiations.

7.2.3 Atom Patterns

This section provides some additional analysis of the generalisations on selected ontologies based on the projection of the regularities in the AD structure. As described in Chapter 6.4, tracing changes in the AD graph after the projection of generalisations, could show patterns that exist across many atoms as a result of combination of generalisations. We name these as *atom patterns.* To give an example, the projection of the syntactic regularities of ontology 73 in the AD causes a 92% compression of the AD structure; this is an indication that the generalisations capture the repetitive structures reflected in the AD as well. In this ontology, the atom pattern consisting of two generalisations is shown in Figure 7.13. This pattern merges 42 atoms, containing axioms instantiating the above two generalisations.

Atom Pattern:
?cluster_4 *SubClassOf* part_of **some** ?cluster_4
?cluster_4 *SubClassOf* ?anatomical_structure

Merged Atoms: 42
Example Atom Instantiations:
Atom 1: ['adult ovary stage II oocyte' *SubClassOf* part_of **some** 'adult ovary',
'adult ovary stage II oocyte' *SubClassOf* 'cell'],
Atom 2: ['adult pharynx cuticular lining' *SubClassOf* part_of **some** 'adult pharynx',
'adult pharynx cuticular lining' *SubClassOf* 'acellular anatomical structure']

Figure 7.13: Dominant atom pattern from ontology 73.

The compressed AD can help to highlight generalisations that are coupled together, giving an intuition of patterns and their deviations as they are projected in the AD. This pattern deviates for two other atoms in the ontology described by the generalisations of Figure 7.14.

Atom Pattern:
(1) ?cluster_4 *SubClassOf* ?anatomical_structure
(2) ?cluster_4 *SubClassOf* part_of **some** ?cluster_4
(3) ?cluster_4 *SubClassOf* part_of **some** ?cluster_3
Merged Atoms: (2)
Atom Instantiations:
Atom 1: [adult hemocyte *SubClassOf* 'cell',
adult hemocyte *SubClassOf* part_of **some** 'adult hemocoel',
adult hemocyte *SubClassOf* part_of **some** 'adult circulatory system']
Atom 2: ['adult sensory neuron' *SubClassOf* 'cell',
'adult sensory neuron' *SubClassOf* 'portion of tissue',
'adult sensory neuron' *SubClassOf* part_of **some** 'adult synganglion',
'adult sensory neuron' *SubClassOf* part_of **some** 'adult peripheral sensillum']

Figure 7.14: A deviation of the atom pattern of Figure 7.13.

It can be observed that the second atom has two axiom instantiations for generalisation (1). Generalisations will reflect both types of repetitive structures; the ones that occur within a single atom and the ones that occur across different atoms. In total, 34 atom patterns were traced in the compressed AD of ontology 73; these patterns occurred across different atoms of the initial AD. This number does not include patterns that could exist in single atoms. For example, an atom with two subclass axioms, can be abstracted by an atom with a single generalisation abstracting both subclass axioms, after the projection of the generalisation on the AD. However, this atom might not be merged with other atoms.

A future direction would be to achieve a better distinction of patterns consisting of more than one generalisations by parsing the connected generalised atoms (top-bottom chain of atoms) in the compressed AD.

7.2.4 Conclusions for Experiment 1

This section described the setup for the experiment for computing syntactic regularities in the BioPortal corpus. It also presented the main analysis of these results. The main outcomes for the syntactic regularities based on the empirical evidence of 93 ontologies are:

- The detection of syntactic regularities by RIO is practical.
- All 93 processed ontologies varying in number of axioms have a form of syntactic regularity covered by at least two clusters.
- The results of clustering using the popularity-based replacement function are comparable to the clustering results when using the structural based replacement function. Clustering using these replacement functions will detect more regularities in the ontologies, which were more coarse-grained (more instantiations per generalisation) compared to the regularities detected by the clustering using a property replacement function.
- In terms of generalisation quality, clustering using the property relevance replacement function performed worse than the other two replacement functions returning fewer generalisations with fewer instantiations per generalisation.
- The detected regularities in the majority of the ontologies caused a compression of the AD (more than 50% for structural and popularity functions). This indicates that the same regularities expressed by the generalisations, also exist in the atoms of the AD. In the cases in which the AD compression was 0%, the reason was that the initial AD consisted of very few atoms with a large number of axioms per atom (more than 50 axioms per atom). Thus, the projection of the generalisations in the AD did not cause any compression across different atoms. It remains as future work to define better metrics for assessing the projection of generalisations in the AD, as an attempt to isolate well defined patterns and isolate irregularities.
- In terms of uniformity of regularities, the majority of big ontologies[6] have at least one generalisation instantiating a significant number of axioms in the ontology. These can be characterised as dominant regularities in the ontology. However, the description of entities that are referenced in dominant regularities can deviate in other axioms in the ontology. This deviation on the remaining axioms is also reflected in the results; each cluster is described by many generalisations with a few instantiations. In other words, there

[6] "Big" ontologies are characterised in this experiment the ontologies that have more than 1000 axioms. This characterisation derives with respect to the collection of the ontologies used in the experiment.

were generalisations abstracting a high number of axioms, but also in the same ontology many more generalisations of smaller abstraction impact.

- Ontologies with highly uniform generalisations had a high number of instantiations and a high cluster coverage. This shows that the clusters are well-formed with respect to meaningful generalisations; generalisations that have many instantiations and cover all the entities of the corresponding cluster.

- Merged atoms, caused by the projection of the generalisations in the AD, can reveal closely coupled generalisations and patterns of more than one generalisation.

7.3 Experiment 2: Qualitative Analysis of Syntactic Regularities

In Experiment 1, we demonstrated the ability of the RIO framework to detect syntactic regularities in a collection of ontologies retrieved from BioPortal. Experiments 2 and 3 show a more in depth analysis of a smaller set of ontologies, for which we have prior knowledge about their construction.

7.3.1 Selected Ontologies

Table 7.4 shows some simple ontology metrics for the five selected ontologies. All the ontologies are documented, enabling further analysis and evaluation of the clusters and regularities.

The Amino Acid ontology[7] has been developed by Robert Stevens, in the School of Computer Science at the University of Manchester. This ontology has been used for teaching purposes. Information about the construction of the ontology can be found online[8].

The Kidney and Urinary Pathway Knowledge Base (KUPKB)[9] ontology [JKS+11] describes:

- The kidney and urinary gross anatomy

[7]`http://goo.gl/WS25H`
[8]`http://goo.gl/O1s57`
[9]`http://129.194.69.119/?q=kupo`

- The cells in those organs and tissues
- The gene products in those cells;
- The functional attributes of those gene products;
- The cellular components of those cells.

The goal of the ontology is to help biologists working on the kidney and urinary pathway to describe biological samples for investigations and the findings from those investigations [JKS$^+$11]. Its project website has a variety of documents describing the construction and the patterns used when developing the ontology.

The Ontology for Biomedical Investigations (OBI) project describes life-science and clinical investigations [BCD$^+$10]. Documentation about the ontology can be found online[10].

SNOMED Clinical Terms (SNOMED-CT) is a large multilingual clinical health terminology[11]. Extensive documentation for the ontology can be found on the online resources of the International Health Terminology Standards Development Organisation (IHTSDO). According to IHTSDO,

> *SNOMED-CT contributes to the improvement of patient care by underpinning the development of Electronic Health Records that record clinical information in ways that enable meaning-based retrieval. This provides effective access to information required for decision support and consistent reporting and analysis.*

Experiment 2 uses two modules from the SNOMED-CT ontology, referring to the description of Hypertension and Chronic clinical findings respectively. More details on the extraction of the modules as well as an extensive analysis of three modules from SNOMED-CT, with regard to its regularities and irregularities, are presented in Experiment 3, Section 7.4. In this experiment we use these two modules from SNOMED-CT to assess the performance of the three replacement methods, the *popularity*, *structural* and *property* replacement functions. In Experiment 3, the method that results in generalisations with the higher abstraction is used for the analysis of the regularities and matching with expected patterns.

[10]`http://obi-ontology.org/`
[11]`http://www.ihtsdo.org/snomed-ct/`

Ontology	**#Axioms**	**#LogicalAxioms**	**#Entities**
Amino acid	381	210	92
SNOMED-CT Hypertension	1 537	492	524
KUBKB	29 406	7 780	3 420
OBI	54 509	28 849	4 027
SNOMED-CT Chronic	13 825	6 957	6 868

Table 7.4: Ontology metrics indicating size of the ontologies used in the qualitative analysis.

Clustering Tasks

The same clustering tasks as described in Experiment 1 were performed with the ontologies in Experiment 2. These are:

1. Clustering using the popularity replacement function.
2. Clustering using the structural replacement function.
3. Clustering using the property relevance function.

7.3.2 Results

Figure 7.15 shows the set of axioms that were instantiating a generalisation. Figure 7.16 shows the mean instantiations per generalisation for all methods. Figure 7.17 shows the cluster coverage of the ontologies for three different methods. For all of the ontologies the mean cluster coverage is more than 20% for at least with one of the methods. Figure 7.18 shows cluster homogeneity for each method and finally Figure 7.19 shows the AD compression for all replacement methods. The Figures show the results for all three replacement methods and the results are sorted by the number of axioms (ontology size) in ascending order.

Atomic Decomposition (AD) Compression. The AD compression in 3 out of 5 ontologies is higher than 50% in at least one of the three replacement methods. OBI has the highest degree of compression (92%) for all three methods; this suggests a very high reliability for the regularities found in the ontology.

Comparison of the three replacement methods. All three variations of the replacement function plugged into the agglomerative hierarchical clustering algorithm have resulted in the detection of regularities in the processed ontologies. In general, there are no high variations between the results of the popularity and structural methods. In Figure 7.15, the percentage of axioms which were

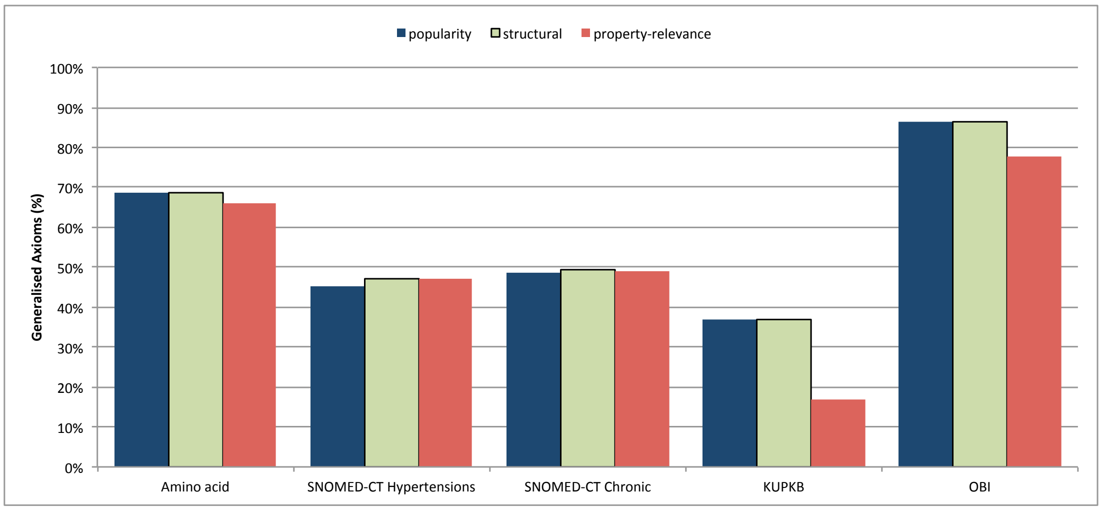


Figure 7.15: Percentage of axioms instantiating a generalisation

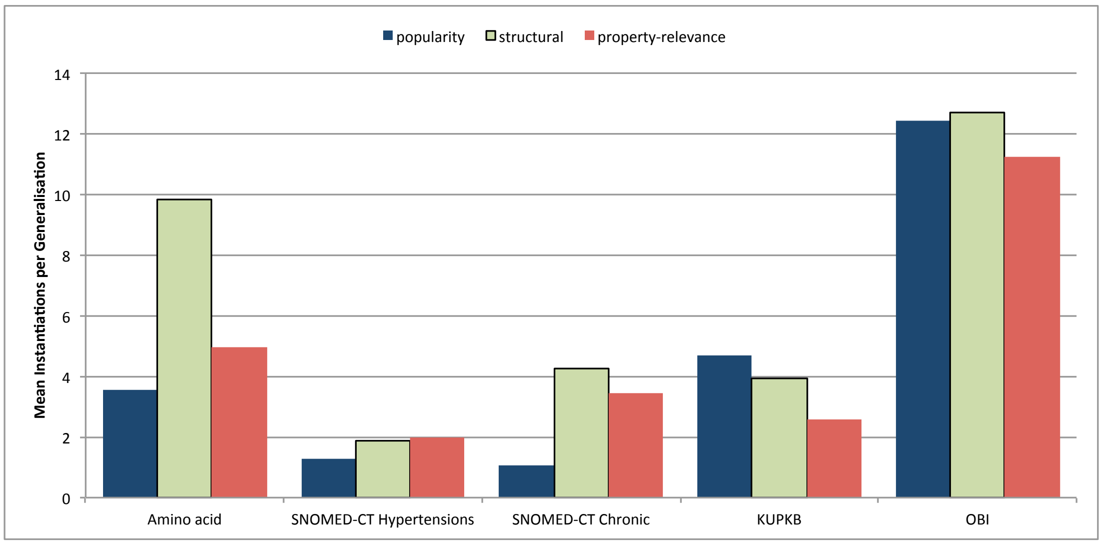


Figure 7.16: Mean Instantiations per Generalisation

found to instantiate a regularity is in most ontologies very close for all methods. The property relevance policy seems to give the smallest number of axioms being generalised compared to the other two methods; however, there is not enough data in this group of ontologies to apply statistical methods to evaluate reliability. It performs the worst in the KUPKB ontology with only 6% of instantiated axioms.

As also depicted in Figure 7.16, the property relevance method will generate generalisations with the lowest number of instantiations compared to the other two methods. Similarly, it gives the lowest AD compression (apart from the

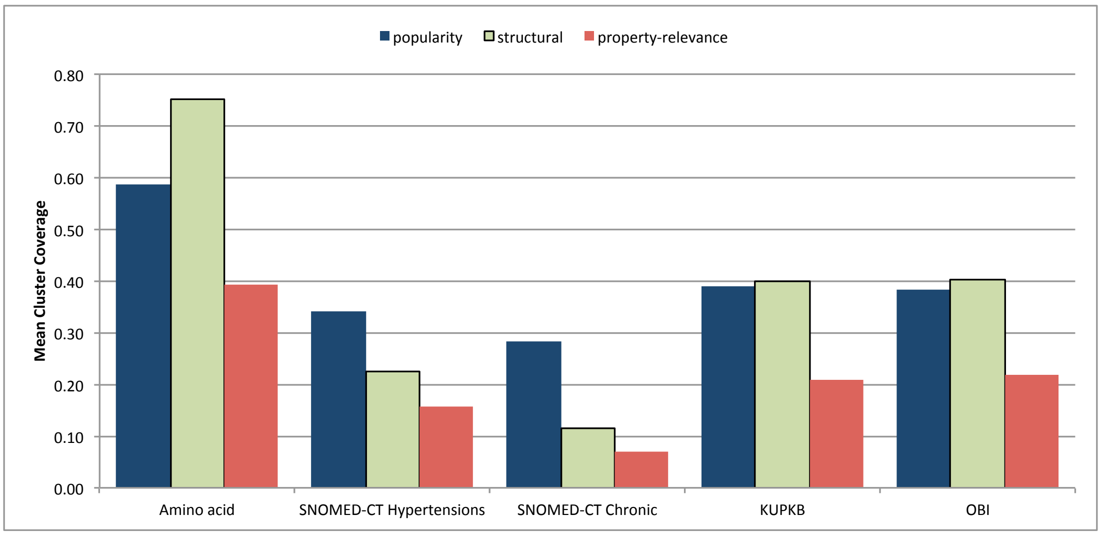


Figure 7.17: Cluster coverage.

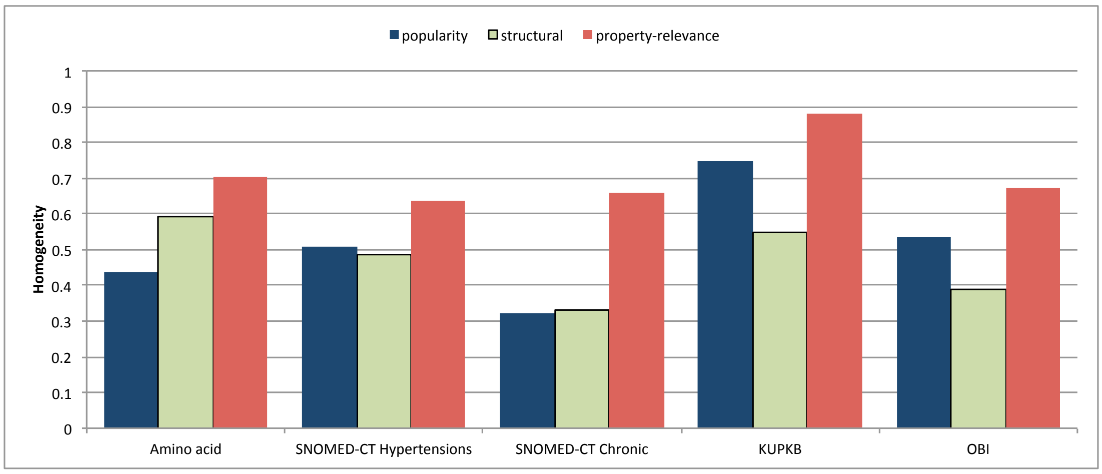


Figure 7.18: Cluster homogeneity

SNOMED-CT Hypertension module) from the other methods. However, property relevance is "winning" in Figure 7.18 as it appears to return the most homogeneous clusters in all ontologies. The reason is that property relevance will produce a very sensitive distance (usually close to value 1) between pairs of entities; thus, fewer entities will be merged into a cluster. As a consequence, fewer axioms will be found to instantiate a generalisation. However, the number of clusters that are formulated have fewer entities and they are more homogeneous. The regularities that are formed capture the most repetitive information in the ontology, but they are quite fine-grained.

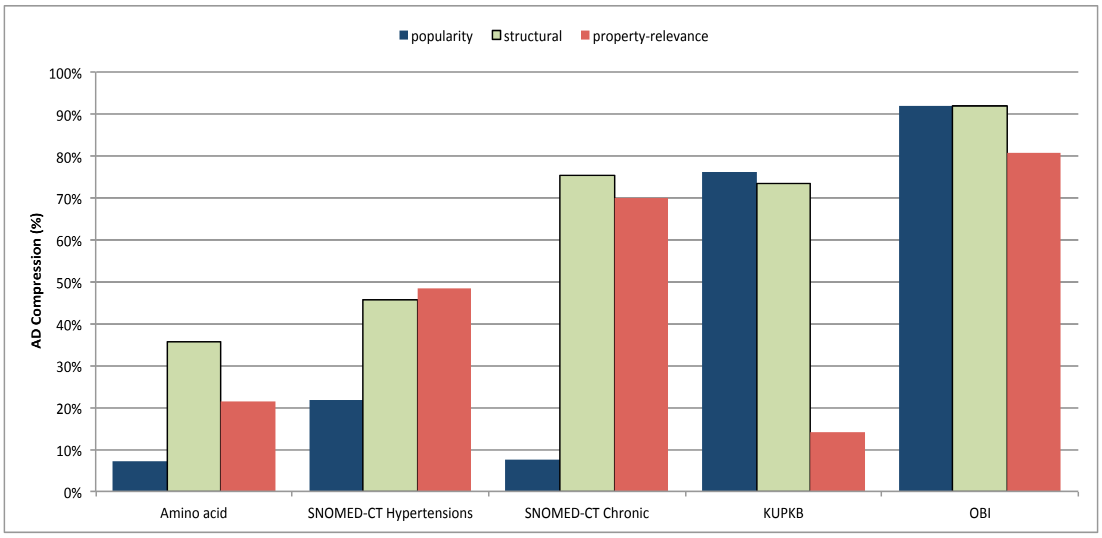


Figure 7.19: Atomic decomposition (AD) compression

On the other hand, the structural replacement method gives better results in the percentage of ontology axioms being generalised, in the abstraction impact of generalisations and in AD compression. However, its cluster homogeneity is the lowest in 3 out of the 5 ontologies. In this case, structural replacement will produce a less sensitive distance, which will lead to the detection of more regularities and with a higher level of abstraction. However, the clusters are not as homogeneous as in property relevance. That is because clusters contain more widely distanced entities.

Finally, the results of the popularity replacement method are close with the ones of the structural replacement method in most cases. There is a big difference only in the Amino Acid ontology and SNOMED-CT modules. In these cases, the percentage of axioms instantiating a generalisation is similar to the one of the structural replacement, however the mean instantiations per generalisation and AD compression are much lower. In general, the popularity replacement method behaves well in most cases. However, sometimes it can overfit the data, meaning that the increased popularity of many entities will have as a consequence a very sensitive distance. The proximity matrix will have more values closer to 1, which will lead to more clusters and very fine-grained generalisations.

7.3.3 Inspection of Regularities

In this section, we will highlight some cases from each ontology. It should be noted that all of the selected ontologies preexisted the clustering framework; this means that insights provided by RIO did not influence the ontology development.

Dominant patterns and verification with documentation

The AminoAcid ontology. The first cluster in the AminoAcid ontology contains all the amino acids. Figure 7.20 shows the main pattern consisting of 7 generalisations. In particular, the pattern describes that every amino acid class is a subclass of the AminoAcid and it has a number of physico-chemical properties. These are the Charge, Polarity, Hydrophobicity, SideChainStructure and Size. This is the main pattern in the ontology, covering 202 axioms, which is 53% of the axioms in the ontology. Such pattern is also described in the online documentation of the ontology[12], named as Entity Property Quality (EPQ) pattern [ERSA08].

The second pattern refers to the asserted structure of the *normalisation* design pattern [AAKS08][13] [Rec03]. The normalisation design pattern is a best practice for avoiding multiple asserted superclasses leading to tangled polyhierarchies. The solution is to assert a primitive class as superclass. Equivalent classes will act as additional categories of the primitive classes. The reasoner will classify the primitive classes under defined categories. Figure 7.21 shows the description of the pattern as it is presented in the online Ontology Design Pattern (ODP) catalogue[14]. The classes Module1, Module2, Module3 are equivalent classes. In the asserted class hierarchy they do not have any subclasses. After the classification of the ontology, the primitive classes are reorganised and have multiple superclasses including the equivalent classes.

In the Amino acid ontology, the Non-PolarAminoAcid whose description is shown in Figure 7.22, is acting as an additional category of the amino acids. In particular, it is a category for the amino acids that are non-polar. In the asserted form of the ontology, the Non-PolarAminoAcid does not have any subclasses. After the classification of the ontology, the Non-PolarAminoAcid will have all amino acids as subclasses which are implied to be non-polar like the Alanine class. RIO will generate a cluster (cluster_5) that includes equivalent classes, acting as categories

[12]http://goo.gl/O1s57
[13]http://ontologydesignpatterns.org/wiki/Submissions:Normalization
[14]http://ontologydesignpatterns.org/wiki/Submissions:Normalization

Generalisations:
(1) ?AminoAcid *SubClassOf* AminoAcid
(2) ?AminoAcid *SubClassOf* ?cluster_3 **some** ?Charge
(3) ?AminoAcid *SubClassOf* ?cluster_3 **some** ?Polarity
(4) ?AminoAcid *SubClassOf* ?cluster_3 **some** ?Hydrophobicity
(5) ?AminoAcid *SubClassOf* ?cluster_3 **some** ?SideChainStructure
(6) ?AminoAcid *SubClassOf* ?cluster_3 **some** ?Size
(7) DisjointClasses: set(?AminoAcid.VALUES)

Example Instantiations:
(1) A *SubClassOf* AminoAcid,
where {?AminoAcid:CLASS=[Alanine]}
(2) A *SubClassOf* hasCharge some Neutral
where {?AminoAcid:CLASS=[Alanine],
?cluster_3:OBJECTPROPERTY=[hasCharge], ?Charge:CLASS=[Neutral]}
(3) A *SubClassOf* hasPolarity some Non-Polar,
where ?cluster_3 = [hasPolarity]
(4) A *SubClassOf* hasHydrophobicity **some** Hydrophobic
where {?AminoAcid:CLASS=[Alanine], ?Hydrophobicity:CLASS=[Hydrophobic],
?cluster_3:OBJECTPROPERTY=[hasHydrophobicity]}
(5) A *SubClassOf* hasSideChainStructure **some** Aliphatic
where {?SideChainStructure:CLASS=[Aliphatic], ?AminoAcid:CLASS=[Alanine],
?cluster_3:OBJECTPROPERTY=[hasSideChainStructure]}
(6) A *SubClassOf* hasSize **some** Tiny
where {?AminoAcid:CLASS=[Alanine],
?cluster_3:OBJECTPROPERTY=[hasSize], ?Size:CLASS=[Tiny]}
(7) *DisjointClasses*: Alanine, Cysteine, Aspartate, Tryptophan, Valine, Threonine, Serine,
Arginine, Glutamine, Proline, Tyrosine, Glycine, Leucine, Methionine,
Asparagine, Histidine, Isoleucine, Lysine
where {?AminoAcid:CLASS=[Alanine, Cysteine, Aspartate, Tryptophan, Valine, Threonine, Serine,
Arginine, Glutamine, Proline, Tyrosine, Glycine, Leucine, Methionine,
Asparagine, Histidine, Isoleucine, Lysine]}

Figure 7.20: Pattern describing the amino acids in the AminoAcid ontology. The pattern consists of 7 generalisations, which describe the amino acids according to their physico-chemical characteristics. An example instantiation of each generalisation is shown for the amino acid Alanine.

in the normalisation pattern. Figure 7.20 shows an example generalisation and instantiation of this cluster.

The KUPKB ontology. In [JHI+10] the design process of the KUP ontology is explained and two main patterns are described for generating the cell types in the ontology. Figure 7.23 shows these patterns described in OPPL [JHI+10].

The results from RIO showed such clusters of cells and clusters of classes used as fillers of the properties describing 'cells' (e.g. participates_in, part_of). Two example generalisations capturing these regularities are shown in Figures 7.24.

Each one of these generalisations corresponds to a different cluster in the ontology. In addition, details corresponding to these regularities was described

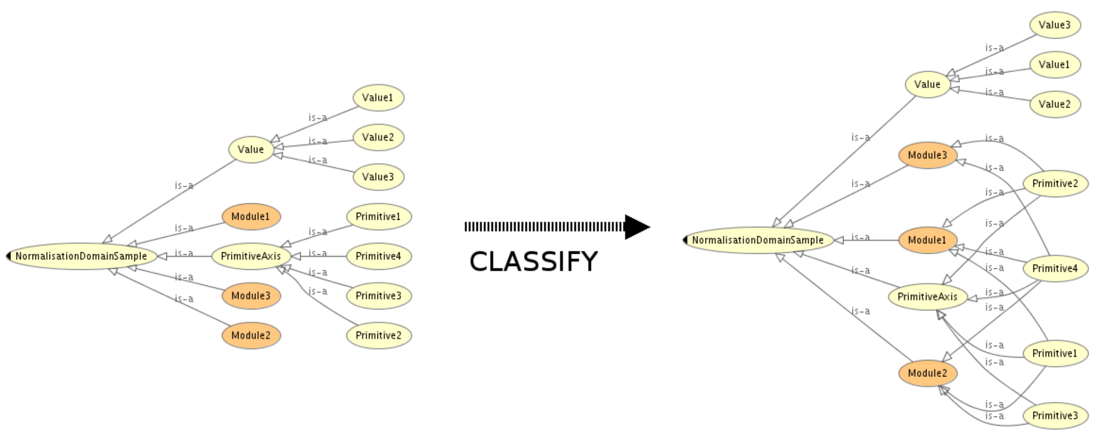


Figure 7.21: Graphical representation of the normalisation pattern. In the asserted class hierarchy the equivalent classes Module1, Module2, Module3 do not have any subclass. After the classification of the ontology, the primitive classes are reorganised and have multiple superclasses.

Generalisation:
?cluster_5 *EquivalentTo* AminoAcid **and** (?cluster_3 **some** ?Polarity)

Example instantiation:
Non-PolarAminoAcid *EquivalentTo* AminoAcid
and (hasPolarity **some** Non-Polar)
where {?cluster_3:OBJECTPROPERTY=[hasPolarity],
?Polarity:CLASS=[Non-Polar], ?cluster_5:CLASS=[Non-PolarAminoAcid]}

Figure 7.22: Example generalisation referring to the normalisation pattern used in the Amino Acid ontology.

in [JHI+10], providing some external validation. The first regularity is encapsulated in the description of the first pattern and the second regularity is encapsulated in the description of the second pattern. Additional regularities were also detected that refer to longer conjunctions of the previous generalisations (e.g. a conjunction of participates_in relationships on the right hand side of the axiom). In this analysis 10 variations of such generalisations were detected.

Pattern 1:

```
?cell:CLASS,
?anatomyPart:CLASS,
?partOfRestriction:CLASS = cell
         and part of some ?anatomyPart,
?anatomyIntersection:CLASS =
   createIntersection(?partOfRestriction.VALUES)
BEGIN
ADD ?cell equivalentTo ?anatomyIntersection
END;
```

Pattern 2:

```
?participant:CLASS,
?participatesRestriction:CLASS = ?cell
        and participates in some ?participant,
?participatesIntersection:CLASS =
      createIntersection(?participatesRestriction.VALUES)
BEGIN
ADD ?cell SubClassOf ?participatesIntersection
END;
```

Figure 7.23: Two OPPL patterns describing cell types in the KUPKB [JHI+10].

The OBI ontology. In [PRG+09], a methodology for developing the 'assay' branch of OBI terms in OBI using a template based on spreadsheets is described. RIO detected 26 clusters including the 'assay' concepts. Figure 7.25 shows the description of such clusters. Cluster ?biological_process includes 54 'epitope specific cells', which are defined classes. It describes the analyte assays that are used to *"achieve a planned objective"*.

The reason that RIO has detected more than one cluster of 'assay' terms is that the asserted axioms describing these terms deviate from a single pattern. For example, cluster ?assay part of whose description is shown in Figure 7.26 includes 53 'assay concepts'. The description of this is different than the one shown in Figure 7.25.

Generalisation:
(1) ?KUPO_core_entity *EquivalentTo* cell
    and (part_of **some** ?KUPO_anatomy_entity)

Example instantiation:
'kidney pelvis cell' *EquivalentTo* cell **and** (part_of **some** 'kidney pelvis')

Generalisation:
(2) ?KUPO_core_entity *SubClassOf* (?cluster_2 **some** 'cytokine production')
    and (?cluster_2 some ?cluster_7)

Example instantiation:
'kidney interstitial fibroblast' *SubClassOf*
    (participates_in **some** 'cytokine production')
    and (participates_in **some**
        'extracellular matrix constituent secretion')

Figure 7.24: Two example generalisations referring to two expected patterns in the ontology. The first generalisation instantiates 60 axioms and the second three instantiations

Generalisations with complete cluster coverage

The generalisations presented in Figures 7.20, 7.22, 7.24 are example syntactic regularities referring to patterns which are expected to be found in the ontologies. In Chapter 4 it was mentioned that a variable in a single generalisation represents a cluster of entities. However, the variable in a particular generalisation does not necessarily cover all members of the corresponding cluster. In many cases it covers a subset of members of a cluster. For example, in the generalisation of Figure 7.25, variable ?biological_process covers 53 out of the 54 members of the cluster.

Alternative view of generalisations

RIO can give an alternative view based on the similar usage of entities in the ontology. It could selectively reveal repeating structures in the ontology that were more difficult to inspect manually. For example, in SNOMED-CT-hypertensions

Generalisation:
?biological_process *EquivalentTo*
?biological_process_1 **and** (?cluster_12 **some** ?biological_4)

Example instantiation:
'epitope specific killing by T cells' *EquivalentTo* 'T cell mediated cytotoxicity'
and ('process is result of' **some** 'MHC:epitope complex binding to TCR').

where
{?biological_process:CLASS=['epitope specific killing by T cells'],
?biological_process_1:CLASS=['T cell mediated cytotoxicity'],
?cluster_12:OBJECTPROPERTY=['process is result of'],
?biological_4:CLASS=['MHC:epitope complex binding to TCR']}

Figure 7.25: An example generalisation of OBI ontology referring to the description of 'assay' classes (cluster ?biological_process). An example instantiation for ?biological_process =['epitope specific killing by T cells'] with corresponding variable replacements is also shown.

Generalisation:
?assay *SubClassOf* ?cluster_2 **some** ?objective_specification

Example instantiation:
'comet assay' *SubClassOf* achieves_planned_objective **some** 'assay objective'

where
{?assay:CLASS=['comet assay'],
?cluster_2:OBJECTPROPERTY=[achieves_planned_objective],
?objective_specification:CLASS=['assay objective']}

Figure 7.26: An example generalisation from the OBI ontology referring to the description of 'assay' classes (cluster ?assay). The generalisation of Figure 7.25 deviates from the generalisation of this figure.

the highlighted classes of Figure 7.27a are grouped in the same cluster and their similar definition is given by the generalisation of Figure 7.27b. However, the inspection of regularities through navigation in the class hierarchy is not an easy task because of the high level of nesting and complexity of the hierarchy. The form of regularity of Figure 7.27b(b) is also described in the technical guide of the ontology [sno11](described in Section 17.2.2., page 180).

7.3.4 Inspecting irregularities

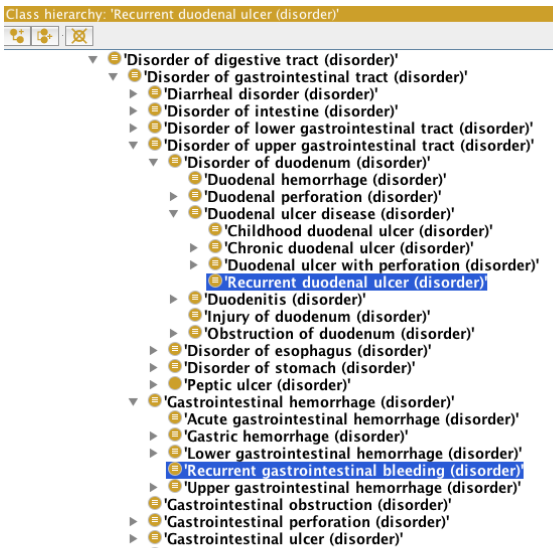
In the example in Figure 7.20, we notice that a possible value of ?AminoAcid is the TinyAromaticAminoAcid. This value is covered only by the third generalisation. The axioms describing this class are shown in Figure 7.28:

TinyAromaticAminoAcid *EquivalentTo* AminoAcid **and** hasSize **some** Tiny
TinyAromaticAminoAcid *SubClassOf* AminoAcid

Figure 7.28: Description of the TinyAromaticAminoAcid in the Amino acid ontology.

The second axiom is redundant causing the TinyAromaticAminoAcid class to be included in the same cluster with the amino acid classes. This cluster is represented with the variable ?AminoAcid. By removing this axiom, the TinyAromaticAminoAcid no longer is a member of cluster ?AminoAcid. This deviation can be characterised as an “anti-pattern”; the redundant subclass axiom had as a result the TinyAromaticAminoAcid to be included in the same cluster with the primitive amino acid classes.

However, there were cases that entities were not included in a cluster because their description was a deliberate exception in the regularity. For example, in OBI, different clusters of ’assay’ terms were detected (described in Figures 7.25 and Figure 7.26).



```
Class hierarchy: 'Recurrent duodenal ulcer (disorder)'
▼ 'Disorder of digestive tract (disorder)'
  ▼ 'Disorder of gastrointestinal tract (disorder)'
    ▶ 'Diarrheal disorder (disorder)'
    ▶ 'Disorder of intestine (disorder)'
    ▶ 'Disorder of lower gastrointestinal tract (disorder)'
    ▼ 'Disorder of upper gastrointestinal tract (disorder)'
      ▼ 'Disorder of duodenum (disorder)'
          'Duodenal hemorrhage (disorder)'
        ▶ 'Duodenal perforation (disorder)'
        ▼ 'Duodenal ulcer disease (disorder)'
            'Childhood duodenal ulcer (disorder)'
          ▶ 'Chronic duodenal ulcer (disorder)'
          ▶ 'Duodenal ulcer with perforation (disorder)'
            'Recurrent duodenal ulcer (disorder)'
        ▶ 'Duodenitis (disorder)'
          'Injury of duodenum (disorder)'
        ▶ 'Obstruction of duodenum (disorder)'
      ▶ 'Disorder of esophagus (disorder)'
      ▶ 'Disorder of stomach (disorder)'
      ▶ 'Peptic ulcer (disorder)'
    ▼ 'Gastrointestinal hemorrhage (disorder)'
        'Acute gastrointestinal hemorrhage (disorder)'
      ▶ 'Gastric hemorrhage (disorder)'
      ▶ 'Lower gastrointestinal hemorrhage (disorder)'
        'Recurrent gastrointestinal bleeding (disorder)'
      ▶ 'Upper gastrointestinal hemorrhage (disorder)'
      'Gastrointestinal obstruction (disorder)'
    ▶ 'Gastrointestinal perforation (disorder)'
    ▶ 'Gastrointestinal ulcer (disorder)'
```

(a) Two members of ?cluster_11 as shown in Protégé class hierarchy view. Although the classes instantiate the same regularity of Figure 7.27b, they are not sibling in the hierarchy.

Generalisation:
?cluster_11 *EquivalentTo* ?cluster_6
and (?cluster_101 **some** (?cluster_57 **some** ?cluster_2))

Example Instantiations:
(1) 'Recurrent duodenal ulcer (disorder)' *EquivalentTo*
'Duodenal ulcer disease (disorder)' **and** (RoleGroup **some**
('Clinical course (attribute)' **some** 'Recurrent (qualifier value)'))
(2) 'Recurrent gastrointestinal bleeding (disorder)' *EquivalentTo*
'Gastrointestinal hemorrhage (disorder)' **and** (RoleGroup **some**
('Clinical course (attribute)' **some** 'Recurrent (qualifier value)')),

where, {?cluster_11:CLASS=['Recurrent duodenal ulcer (disorder)', 'Recurrent gastrointestinal bleeding (disorder)'], ?cluster_6:CLASS=['Duodenal ulcer disease (disorder)', 'Gastrointestinal hemorrhage (disorder)'], ?cluster_101:OBJECTPROPERTY=[RoleGroup], ?cluster_57:CLASS=['Clinical course (attribute)'],
?cluster_2:CLASS=['Recurrent (qualifier value)']}

(b) Generalisation and two example instantiations from SNOMED-CT-chronic module.

Figure 7.27: Syntactic regularities are not detected only between sibling classes. An example regularity is shown in Figure 7.27b. The position of cluster_11 =['Recurrent duodenal ulcer (disorder)', 'Recurrent gastrointestinal bleeding (disorder)'] in the class hierarchy is shown in Figure 7.27a.

7.3.5 Summary for Experiment 2

In this Experiment we showed the ability of RIO to represent patterns amongst the detected syntactic regularities. For each of the selected ontologies we demonstrated a corresponding regularity that captures an expected pattern. The next section has an extensive analysis of the syntactic regularities on three modules of the SNOMED-CT ontology; the aim of the following Experiment is to perform a more systematic *quality assurance* of the SNOMED-CT modules with respect to their underlying patterns.

7.4 Experiment 3: Analysis of Syntactic Regularities in SNOMED-CT

In this Section the pattern based quality assurance of three modules [DTI07] from SNOMED-CT is presented. In particular, we detect and further analyse the regularities and irregularities in the modules of the ontology, and find how these can be linked to potential design defects in the ontology that have been reported in past work. The assumption made for these defects is that entities that follow naming conventions should also follow a similar pattern in the description of their usage axioms. For example, any concept in the ontology that is labeled as Chronic, should also have an explicit or implicit reference to the 'Chronic (qualifier value)' class. Entities that do not follow this pattern are categorised as:

1. Design discrepancies in the asserted axioms in an ontology.
2. Deliberate deviations of a pattern

We pinpoint such defects in the ontology and we verify a portion of them by referring to the SNOMED-CT literature. The design discrepancies we highlight mainly refer to missing restrictions. The rest are categorised as deviations from an expected pattern. We show that parts of the ontology that do not follow a particular pattern are more prone to design discrepancies, such as missing restrictions, incorrect descriptions etc. That is expected, since the developers have a higher level of freedom to describe concepts that do not have a general pattern, and, therefore, there is more room for error.

7.4.1 SNOMED-CT modules

The steps that were followed for the analysis of the regularities in SNOMED-CT were:

1. Extraction of three SNOMED-CT modules
2. Application of the RIO on the asserted ontologies
3. Analysis of the regularities and verification with published work and SNOMED-CT documentation.

For the extraction of the modules we used the July 31, 2010 IHTSDO (International Health Terminology Standards Development Organisation) release of SNOMED CT converted to OWL using the Perl script provided with the release.

The procedure for extracting the modules from the ontology is the same as the one described in [RI12]. For module extraction, we used the methods in the OWL API[15] [HB09], packaged in an application available online [16].

The analysis of syntactic regularities mainly focuses on the description of four groups of terms in the ontology. These groups of terms have a naming convention and it is also expected to instantiate a pattern. In the remainder of the Chapter it will be called a "target" entity. We also will refer to terms in the ontology by using their labels as they were found in the OWL ontology. The extracted modules refer to "chronic" and "acute" diseases, together with "present" and "absent" clinical findings. Some descriptive metrics for the size of the modules are presented in Table 7.5.

	Present and absent clinical findings	Chronic findings	Acute findings
Target entities	Classes whose labels have the keywords "present" or "absent"	Classes whose labels have the keywords "chronic"	Classes whose labels have the keywords "acute"
Axioms	5 065	20 688	19 812
Classes	1 687	6 842	6 599
Object properties	16	25	25
Mean class hierarchy depth	9.76	11.2	10.09

Table 7.5: General metrics on the three extracted modules of SNOMED-CT.

The extraction of the modules was based on a set of terms that were expected to instantiate a pattern in their axioms. For example, it is expected that all

[15]http://owlapi.sourceforge.net

[16]http://owl.cs.manchester.ac.uk/research/topics/snomed/

chronic findings have an axiom that relates them with 'Chronic (qualifier value)'. Similarly all acute findings are expected to be related with 'Acute (qualifier value)'. The modules describe classes that are findings and have the words "Present" or "Absent" in the beginning or middle of their label. These can be gathered using the OPPL script in Figure 7.29.

```
1.
?c:CLASS, ?x:CONSTANT = MATCH(".Present.*")
SELECT  ?c.IRI label ?x
BEGIN
ADD ?c subClassOf PresentInTheBeginningCandidate
END;

2.
?c:CLASS, ?x:CONSTANT = MATCH(".*present.*")
SELECT  ?c.IRI label ?x
BEGIN
ADD ?c subClassOf PresentInTheMiddleCandidate
END;
```

Figure 7.29: OPPL scripts for gathering target terms for the extraction of the "Present" module

The OPPL script gives 0 candidate classes whose name starts with the word "Present" and 59 classes with the word "present" in the middle of their name. For the "absent" case, there was only 1 class whose label started with the word "Absent" and 24 classes with the word "absent" in the middle of their name.

Similarly, the "Acute" and "Chronic" modules were extracted based on a set of terms which have the words "Acute" and "Chronic" in their name and describe acute and chronic clinical findings respectively. Similar scripts as the ones presented in Figure 7.29 gave 420 candidate "Chronic" classes and 509 "Acute" candidate classes.

In our analysis the patterns which are described in the technical guide [sno11] are manually formulated using OPPL scripts, which returns all possible candidate axioms that are instantiations of this pattern. These are compared with RIO's results and we further discuss the strengths and weaknesses of the two methods.

7.4.2 Analysis of the syntactic regularities

Table 7.6 show some evaluation metrics of the application of the RIO framework on each module. As it has been already described in Chapter 6, the mean number of instantiations per generalisation shows how many axioms can be abstracted by a single generalisation. An ontology with a high cluster coverage per generalisation (close to 100%) and a high mean of instantiations per generalisation is a strong indication of a very regular ontology.

Module	# Clusters	Cluster coverage per generalisation (%)	Mean Instantiations per Generalisation
Present and absent clinical findings	41	8.50	6.42
Chronic findings	75	6.40	6.13
Acute findings	76	6.80	5.70

Table 7.6: Results of the application of RIO in the three SNOMED-CT modules.

The results show that the present and absent clinical findings module is the most regular of all three modules, as it has the highest cluster coverage and mean instantiations per generalisations. However, the overall cluster coverage percentage does not exceed 8.5%. There are two reasons for this result; First, the ontology is not highly regular. Secondly, in some cases the clustering algorithm is too "greedy", resulting in big clusters that are not completely homogeneous.

Most of the regularities that were captured by RIO refer to restrictions using the RoleGroup attribute [SDMW02] for grouping relationships. This is also the main regularity that is described in the technical guide of the release we used [sno11]. The purpose of the RoleGroup attribute in SNOMED-CT was to provide a simple way to indicate that certain roles should be grouped together [SDMW02]. However, we want to check how this general regularity is formed when describing different sets of terms in the ontology. In the remainder of the section, we will focus on the regularities we found in the entities from the ontology, which have:

- The words "Present" or "Absent" at the beginning or in the middle of their name.
- The words "Chronic" or "Acute" at the beginning or in the middle of their name.

7.4.3 "Present" and "Absent" cases

Table 7.7 shows the results of the regularities referring to entities, whose names included the words "present" or "absent". There are 58 out of 59 "present" entities which are distributed in 6 clusters. Similarly, all the 25 "absent" entities are distributed in 5 clusters.

There is one "present" clinical finding that is not included in the clusters ('Definitely present (qualifier value)'). The reason is that this class is never used in any other axiom, apart from its declaration and position in the class hierarchy (It is a subclass of the 'Known present (qualifier value)' and superclass of the 'Confirmed present (qualifier value)'. Fifty of the "present" entities appear to be in the same cluster and described by 16 generalisations. Figure 7.30 shows an example generalisation and instantiation referring to this cluster (cluster_1).

The structure of the 16 generalisations describing "present" classes is similar; in many cases the only thing that changes is a single variable in the generalisation. A reason for this is that entities participating in axioms of similar syntax fall into different clusters due to their different usage in other axioms. The expected pattern for all the present and absent cases is their explicit reference to the 'Known present (qualifier value)' and 'Known absent (qualifier value)' respectively. An example instantiation pattern is shown in Figure 7.30.

The analysis of the regularities (Table 7.7) showed that the 31% (127) of the usage axioms of the present entities are using this pattern. These axioms are abstracted by 23 (35%) of the generalisations. Similarly, for the absent classes, 81 (34%) instantiations explicitly refer to the absent qualifier value and these are abstracted by 15 (39%) by the generalisations. Note that the total number of instantiations in Table 7.7 refer to all the axioms that were generalised and referenced by at least one target entity (including both left and right hand side of the axiom). Also these numbers refer to the expected syntactic pattern, which is an explicit reference to a qualifier value (e.g. 'Known present (qualifier value)'). However, an implicit pattern can exist, such as propagation through the class hierarchy, which can infer such a connection. This explains the relatively low percentage of the axioms following the pattern.

```
Generalisation:
?cluster_1 EquivalentTo ?cluster_20
          and ?cluster_23 some ?expression_conjunction
             and ?cluster_23 some ?cluster_16

Example Instantiation:
'Coin sign present (situation)' EquivalentTo
       'Clinical finding present (situation)'
          and (RoleGroup some ((Associated finding (attribute)
               some 'Coin sign (finding)')
          and ('Finding context (attribute)'
               some 'Known present (qualifier value)')
          and ('Temporal context (attribute)'
            some 'Current or specified time (qualifier value)')
          and ('Subject relationship context (attribute)'
            some 'Subject of record (person)')))
where:
?cluster_1: CLASS=['Coin sign present (situation)'],
?cluster_20: CLASS=['Clinical finding present (situation)'],
?cluster_23: OBJECTPROPERTY=[RoleGroup, Subject relationship context
(attribute)], ? expression_conjunction: CLASS=[
     ((Associated finding (attribute) some Coin sign (finding))
      and (Finding context (attribute) some Known present (qualifier value))
      and (Temporal context (attribute) some Current or specified time (qualifier value))],
?cluster_16: CLASS=['Subject of record (person)']
```

Figure 7.30: Example generalisation and instantiation for ?cluster_1. The cluster includes 50 classes with the word "present" in their name described by 16 generalisations. The example generalisation and instantiation show the pattern that is used for describing these entities, which is the usage of particular roles.

Similarly, 21 of the "absent" clinical findings are in the same cluster described by 10 generalisations. Both *absent* and *present* classes seem to follow the same type of regularity in their definition. This is also described as a common pattern in the SNOMED-CT online resources [sno11]. Most of the entities are described using the RoleGroup attribute for grouping relationships using the 'Associated finding (attribute)', 'Finding context (attribute)', 'Temporal context (attribute)' and 'Subject relationship context (attribute) attributes. However, in both cases, we found deviations from this pattern.

There were also clusters including entities with different forms of class expressions. An example is the 'Clinical finding absent (situation)'. The analysis of the module gave in total 17 classes that included the 'RoleGroup' attribute more than once in their definition. Figure 7.31 shows three example clusters with such entities. The usage of role groups and multiple role groups in SNOMED-CT is described in [CS09].

These examples might be deliberately defined in this way, however it is a deviation from the design style of most classes, that leads to deeply nested class expressions. It might be also a case of malformed axioms, as it is not clear in which cases the 'RoleGroup' attribute should be used more than once in the same axiom [CS09] and when a relationship should be grouped with an existing role group. Table 7.7 shows that 3 clusters were detected with 4 "present" classes using multiple role groups in their axioms. Likewise, 3 clusters were detected with 3 "absent" classes whose axioms used multiple role groups. Our aim is to highlight such cases, which should be further assessed by experts.

?cluster_19:{
'Foreign body in female genital organs
and perineum (disorder)'
'On examination - diabetic maculopathy
absent both eyes (situation)'
'Bilateral cataracts (disorder)'
'Alexia and agraphia present (situation)'
'Foreign body of body cavity and wall (disorder)'
'Injury to heart and lung (disorder)'
'Tumor of lower respiratory tractand
mediastinum (disorder)'
'On examination - diabetic maculopathy
present both eyes (situation)'}

?cluster_20:{
'Musculoskeletal and connective tissue
disorder (disorder)'
'Finding with explicit context (situation)'
'History of clinical finding in subject (situation)'
'Clinical finding present (situation)'
'Clinical finding absent (situation)'
'Disorder of soft tissue of thoracic
cavity (disorder)'}

?cluster_28:{
'On examination - genitalia (finding)'
'On examination - skin (finding)'
'Disorder of soft tissue of body cavity (disorder)'}

(a) Outlier clusters with multiple usage of RoleGroup attribute in their axioms

Clinical finding absent (situation) *EquivalentTo*
Situation with explicit context (situation) **and**
(RoleGroup **some** (Finding context (attribute)
some Known absent (qualifier value)))
and (RoleGroup **some**
(Temporal context (attribute)
some Current or specified time (qualifier value)))
and (RoleGroup **some**
(Subject relationship context (attribute)
some Subject of record (person)))

(b) Example axiom with multiple usage of the RoleGroup attribute

Figure 7.31: Clusters with multiple usage of the RoleGroup attribute and example instantiation.

	Present	Absent
Total number of entities starting with "Present" or "Absent"	0	1
Total number of entities having "present" or "absent" in the middle of their name	59	24
Number of clusters that include the target entities	6	5
Number of generalisations describing the target entities	65	39
Number of instantiations referring to the target entities	404	236
Number of target entities that were not in any cluster.	1	0
Number of clusters including entities with multiple role groups (RoleGroup) in their axioms	3	3
Number of clustered entities using multiple role groups (RoleGroup) in their axioms	4	3
Number of generalisations which instantiations explicitly refer to the present (Known present (qualifier value)) or absent qualifier (Known absent (qualifier value))	23 (35%)	15 (39%)
Number of instantiations that explicitly refer to the present or absent qualifier	127 (31%)	81 (34%)

Table 7.7: Selected results of the analysis of regularities in present and absent cases.

7.4.4 "Chronic" and "Acute" cases

Table 7.8 summarises the results of the regularities that were found in the entities containing the words "acute" and "chronic" in their label. The results showed that most of the entities do not follow a general pattern. Therefore, the entities are distributed in many clusters and are described by many generalisations. From the technical guide, a general pattern that is expected in these terms is the explicit reference to the chronic or acute qualifiers in equivalent or subclass axioms [RI12]. An example description is shown in Figure 7.32.

'Chronic urate nephropathy (disorder)' *SubClassOf* 'Urate nephropathy (disorder)'
  and (RoleGroup **some** ('Clinical course (attribute)' **some** 'Chronic (qualifier value)'))

Figure 7.32: Example description of a "chronic" class ('Chronic urate nephropathy (disorder)').

However, only 50 (5%) of the generalisations for the "Chronic" module were found to abstract axioms related to "Chronic" entities and 114 (10%) of the generalisations for the "Acute" module abstracted axioms related to "Acute" entities. An example generalisation reflecting the expected pattern for the chronic classes is shown in Figure 7.33.

We verified some of the results of this work with the results described in [RI12]. However, this chapter focuses more on the analysis of the syntactic regularities from an ontology engineering perspective. Subsets of these terms might be expected to deviate from this pattern from a medical perspective. For example, a subset of "chronic" terms deviate from the pattern in Figure 7.33 reported in [RI12], as they are described according to their morphology. Thus, there is no existential restriction in their asserted axioms referring to the 'Chronic (qualifier value)'. Since these terms do not have a reference to the Chronic (qualifier value) they cannot be highlighted by a syntactic tool like RIO; such a relationship is found in the inferences of the ontology.

Generalisation:
?cluster_2 *EquivalentTo* ?cluster_12 **and** (RoleGroup **some**
(?cluster_32 **some** ?cluster_20))

Example instantiation:
'Chronic pyonephrosis (disorder)' *EquivalentTo* 'Pyonephrosis (disorder)'
and (RoleGroup **some** ('Clinical course (attribute)' **some** 'Chronic (qualifier value)'))

where:
{?cluster_2:CLASS=['Chronic pyonephrosis (disorder)'],
?cluster_12:CLASS=['Pyonephrosis (disorder)'],
?cluster_32:OBJECTPROPERTY=[Clinical course (attribute)],
?cluster_20:CLASS=['Chronic (qualifier value)']}

Figure 7.33: Example syntactic regularity that covers 14 axioms describing 14 chronic disorders. This syntactic regularity reflects a pattern that expected to be found for "chronic" classes (explicit reference to the 'Chronic (qualifier value)').

In addition, 20 entities from the target set of entities were not clustered. Here, for the sake of brevity, we mainly focus on the "Chronic" cases. Table 7.8 summarises some of the results for all the cases. Figures 7.34 and 7.35 show the chronic and acute entities, which were not included in any cluster respectively. Some of them, such as the 'Chronic anxiety (finding)' were also reported in [RI12] as design defects. From these, the "Chronic low back pain (finding)" is reported in [RI12] as having an incomplete description. In particular, the class has an existential restriction that is missing. However, this type of irregularity is not clear from the syntax of the axiom. The class is grouped with other "chronic" classes, which have complete existential restrictions. Therefore, this type of irregularity is a challenge to identify using the analysis of the results.

	Chronic	**Acute**
Total number of entities starting with "Chronic" or "Acute"	388	472
Total number of entities having "chronic" or "acute" in the middle of their name	32	38
Number of clusters that include the target entities	34	34
Number of generalisations describing the target entities	919	1109
Number of instantiations referring to the target entities	1503	1849
Number of target entities that were not in any cluster.	12	11
Number of clusters including entities with multiple role groups (RoleGroup) in their axioms	19	21
Number of clustered entities using multiple role groups (RoleGroup) in their axioms	64	79
Number of generalisations which instantiations explicitly refer to the chronic (Chronic (qualifier value)) or acute qualifier (Sudden onset AND/OR short duration (qualifier value))	50 (5%)	114 (10%)
Number of instantiations that explicitly refer to the chronic or acute qualifier	76 (5%)	210 (11%)

Table 7.8: Selected results on the analysis of regularities in chronic and acute cases.

'Chronic progressive renal failure (disorder)',
'Chronic back pain (finding)',
'Chronic diarrhea of unknown origin (disorder)',
'Chronic inflammatory demyelinating polyneuritis (disorder)',
'Chronic cough (finding)', 'Chronic anxiety (finding)',
'Chronic post-traumatic stress disorder (disorder)',
'Chronic bullous emphysema (disorder)',
'Chronic acquired lymphedema (disorder)',
'Chronic diarrhea (disorder)', 'Chronic constipation (disorder)',
'Chronic pain syndrome (disorder)'

Figure 7.34: Chronic entities that were not included in a cluster.

'Acute exacerbation of bronchiectasis (disorder)',
'Acute cardiac pulmonary edema (disorder)',
'Acute coronary syndrome (disorder)',
'Acute and subacute liver necrosis (disorder)',
'Acute myeloid leukemia with recurrent genetic abnormality (morphologic abnormality)',
'Gallbladder calculus with acute cholecystitis and no obstruction (disorder)',
'Acute situational disturbance (disorder), Acute urticaria (disorder)',
'Acute meniscal tear, medial (disorder)',
'Acute contagious conjunctivitis (disorder)',
'Acute schizophrenia-like psychotic disorder (disorder)'

Figure 7.35: Acute entities that were not included in a cluster.

Finally, 10 clusters included entities participating in nested class expressions with multiple role groups (using multiple RoleGroup relationships). It should be noted that clusters of a smaller size tended to include such deviations from the regular pattern (most of the class expressions are described using a single role group in the examined modules).

7.4.5 Verification of results

In order to verify some of the results, we manually ran OPPL scripts whenever possible and estimate recall on the regularities. In particular, we examined which classes failed to follow the expected pattern by running corresponding OPPL scripts. For example, for the "chronic" classes we ran the queries of Figure 7.36 to select classes that had the word "Chronic" in their label, but were missing the expected semantics. The first OPPL scripts of Figure 7.36 gave 131 candidate classes with incomplete semantics while the second script will give 5 candidate classes with incomplete semantics. This set of candidate classes are potential errors in the ontology since they are missing the reason for being "chronic" findings, despite this is indicated in their label.

```
1.
?c:CLASS, ?x:CONSTANT=MATCH(".Chronic.*")
SELECT  ?c.IRI label ?x
WHERE FAIL  ?c subClassOf RoleGroup some
 ('Clinical course (attribute)' some 'Chronic (qualifier value)')
BEGIN ADD ?c subClassOf ChronicIncompleteCandidates
END;

2.
?c:CLASS, ?x:CONSTANT=MATCH(".*chronic.*") SELECT  ?c.IRI label ?x
WHERE FAIL
?c subClassOf RoleGroup some
  ('Clinical course (attribute)' some 'Chronic (qualifier value)')
BEGIN ADD ?c subClassOf ChronicIncompleteCandidates END;
```

Figure 7.36: OPPL scripts for gathering chronic classes with incomplete description.

Comparing the results of the manual analysis using OPPL scripts with the results of the automatic analysis by RIO we can note that the analysis with RIO gave in total 314 classes (Table 7.8) as potential deviations from the expected pattern while manual OPPL scripts narrowed this down to 131 candidate classes.

The reason for this difference is that OPPL scripts take into account both the asserted and the inferred form of the ontology, thus the instantiation of the expected pattern is in the inferences of the ontology. However, this kind of semantic analysis could not be done by RIO since we have a purely syntactic approach. It should be mentioned, though, that the 131 candidate classes from the OPPL script included all these classes in Figure 7.34. Similar OPPL scripts gave 147 "acute", 9 "present" and 2 "absent" candidate classes with missing descriptions.

7.4.6 Summary for Experiment 3

In this experiment we presented a more in depth analysis of syntactic regularities in three modules of SNOMED-CT ontology with respect to patterns that were expected to be instantiated by terms with particular keywords in their labels (referring to them as "target" entities).

The results showed expected regularities, as well as deviations from these regularities. It revealed terms with incomplete descriptions, such as missing existential restrictions (e.g. 12 "chronic" classes with an incomplete description which were not included in any cluster); classes, which were placed correctly in the class hierarchy by the reasoner, but described with long class expressions using multiple role groups (e.g. 79 "acute" classes whose definition makes use of multiple role groups). In the worst case, the expected patterns described in the technical guide of the ontology were explicitly instantiated by only 5% of the corresponding entities in the module (Table 7.8). The results also indicated that parts of the ontology that did not follow an explicit pattern tended to have more potential "defects". All these can be detected and reported to domain experts who will decide which ones should be modified.

In addition, with OPPL scripts (Figure 7.36) we verified a subset of entities which were also highlighted with the analysis from RIO (e.g. entities of Figure 7.34). However, a number of target entities did not instantiate the expected pattern in their asserted axioms, but the pattern was expressed in the inferences of the ontology. These entities were highlighted by RIO as deviations from the expected pattern as the inferences were not taken into account. The consideration of inferences by RIO for detecting patterns in inferences of the modules is presented in the next Chapter.

Chapter 8

Experiments on Semantic Regularities

We have presented so far the experiments on the analysis on the syntactic regularities in ontologies. In this section we present the corresponding analysis for semantic regularities of ontologies. Again, the outline of the analysis is very similar to the one used for the syntactic regularities.

8.1 Experiment 4: BioPortal Semantic Regularities

Similar to Experiment 1, in Section 7.2, this section describes the quantitative analysis of the detection of semantic regularities in the BioPortal repository. We used the same collection of BioPortal Ontologies as in Experiment 1. Initially we will describe the ontologies used for the experiment and then the presentation and discussion of the results. The setup of the experiment and the presentation of the results were described in Chapter 7.1.

8.1.1 Results

With the timeout considerations, RIO completed all three semantic clustering tasks for a set of 40 ontologies. This section presents the results of these tasks and a discussion with additional analysis of the results follows.

Table 8.1 shows the total mean values for selected metrics of the processed BioPortal ontologies while Figures 8.1, 8.2, 8.5 present the semantic regularity results for each ontology for all three replacement methods. In these figures, the results are sorted in ascending order according to the number of axioms of the ontologies. As it is indicated in the x-axis of the Figures, the ontologies were categorised according to their axiom number to *small ontologies* (30-300 axioms), *medium ontologies* (300-1000 axioms) and *big ontologies* (1000-12000 axioms).

Metrics	**Popularity**	**Structural**	**Property Relevance**
Generalised Entailments (%)	100%	39%	100%
Instantiations per Generalisation	97.5	27.5	24.6
# Clusters	4.4	2.85	5.4
# Entities per Cluster	136.1	69.5	41.7
Cluster Coverage (%)	48%	32%	41%
Homogeneity	0.40	0.65	0.65
AD Compression (%)	81%	41%	40%

Table 8.1: Total mean values of semantic regularity metric results for the 40 processed ontologies from the BioPortal repository.

According to Table 8.1, the mean instantiations per generalisation value indicates that the clustering based on structural and popularity replacements functions gives generalisations of higher abstraction. These generalisations also correspond to repetitive structures reflected in the Atomic Decomposition as the AD compression in Table 8.1 is higher for popularity and structural replacement function. The information depicted in Figures 8.2, 8.5 seems to be in an agreement with the initial hypothesis. However, more information is needed to give a clear answer. Further discussion and verification of these results are presented in the following section.

8.1.2 Discussion

Detection of regularities

The practicality of computing semantic regularities. The computation of semantic regularities for all three methods was completed in the allotted time for 40 out of 208 BioPortal ontologies. The performance for the computation of semantic regularities is worse than the one for the computation of syntactic regularities; in fact, there are two bottlenecks in the computation of semantic regularities; the extraction of entailments from the KE and the computation of the proximity matrix for the clustering algorithm.

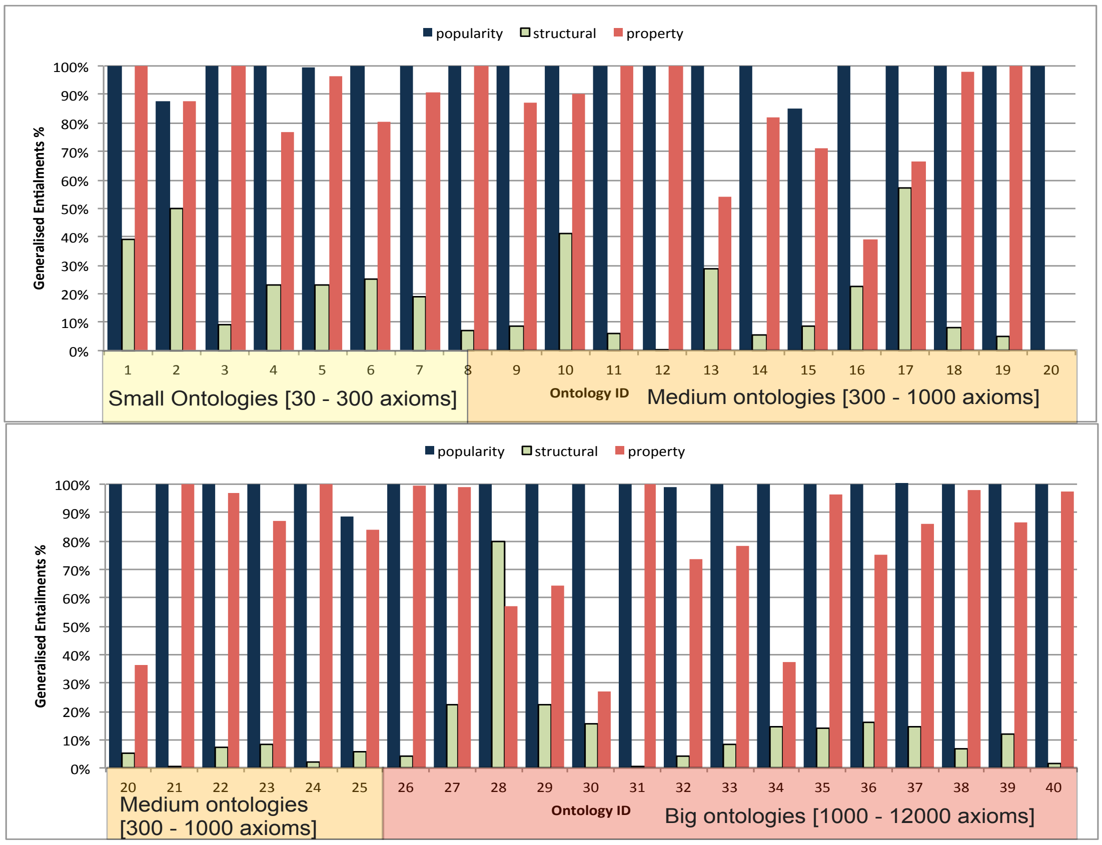


Figure 8.1: Percentage of generalised entailments in BioPortal ontologies

KE is already an optimisation for quicker computation of entailments, but it still requires reasoning and is therefore an onerous task. Chainsaw[1], an experimental reasoner that leverages Atomic Decomposition, was firstly motivated by this problem, i.e., how to tackle reasoning over very large ontologies that can be decomposed with AD [TP12]. However, it is still in the experimental stage, so it was not used extensively in these experiments.

The proximity matrix computation is not a bottleneck unique to semantic regularities; the same problem affects syntactic regularities, to a lesser extent as on average the matrices to be computed are smaller in the syntactic approach. Even though various optimisations have been applied to the computation of the proximity matrix and the clustering procedure, more can be done to allow RIO to better handle ontologies consisting of hundreds of thousands of axioms or entailments; such improvements have been planned for future work.

[1]`http://chainsaw.sourceforge.net`

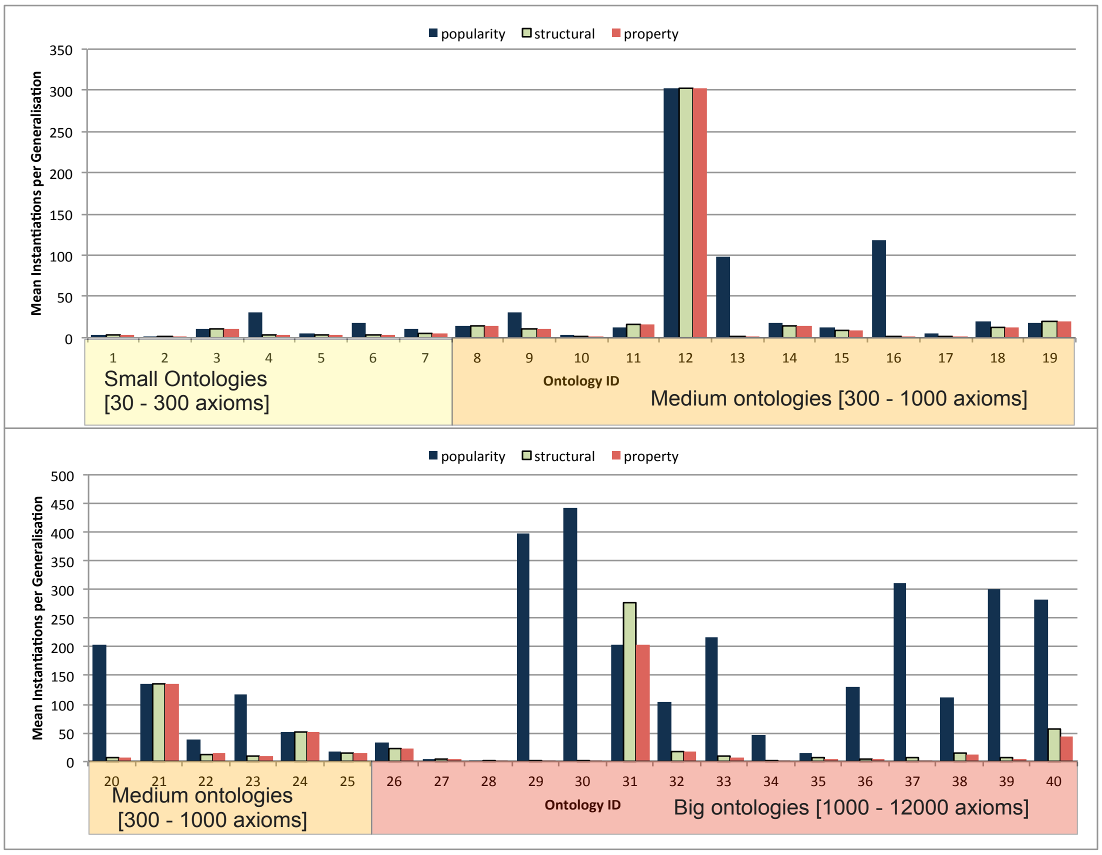


Figure 8.2: Mean Instantiation per Generalisation of BioPortal Ontologies.

It is nonetheless an acceptable performance to detect semantic regularities for all clustering methods in less than 45 minutes. The largest ontology (bioinformatics-data-formats-identifiers-operations-and-topics) that was processed with this time-out consisted of maximum 20 951 entailments. It should be also noted that the number of entailments is not directly comparable with the number of axioms in the ontology, i.e., it is not possible, before applying a reasoner, to know what this number will be. For example, in the set of processed ontologies, the ontology with the maximum number of entailments was not the one with the largest set of asserted axioms. It is expected though, for all of the ontologies, the number of entailments of each ontology will be greater than the number of its asserted axioms. That is because the KE will extract both trivial and non-trivial entailments from the ontology.

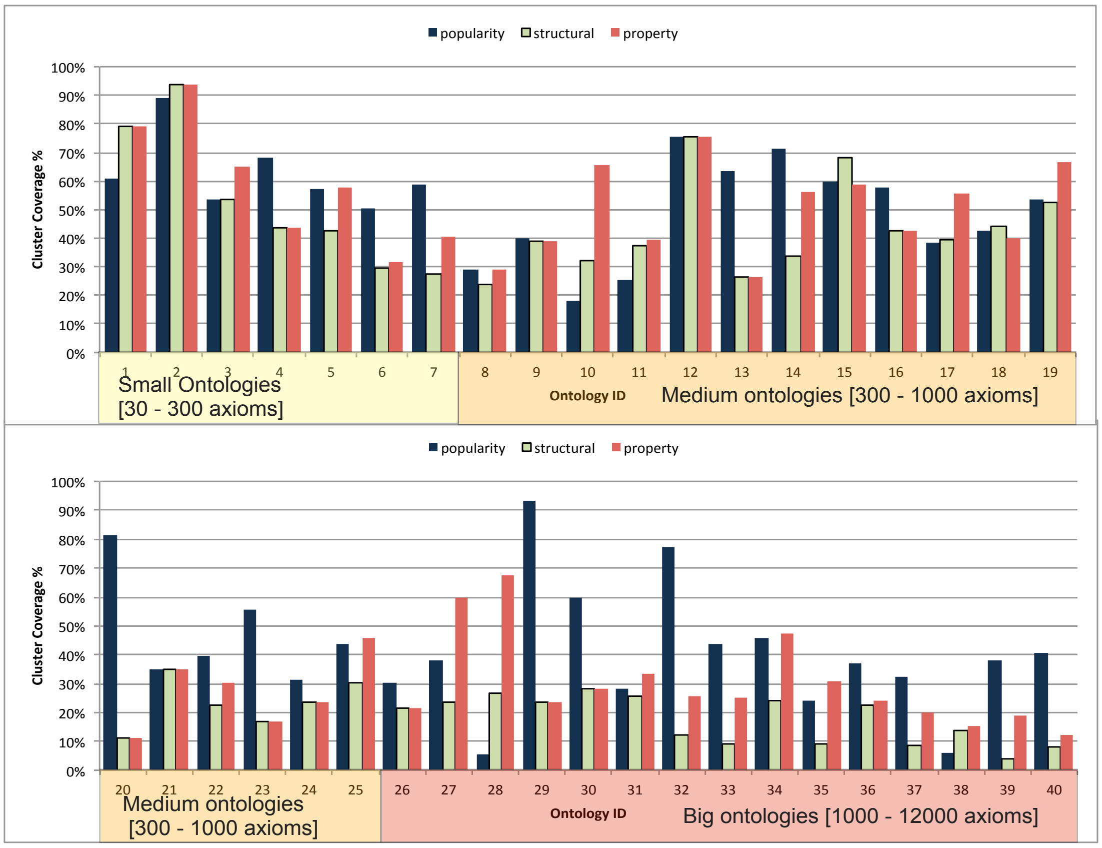


Figure 8.3: Mean Cluster Coverage of BioPortal Ontologies.

Comparison of replacement methods

Significance of Differences. Table 8.2 shows the p value for the Student's t-Test revealing the significant difference on metric values for different replacement methods[2]. Each column in the table represents a potential hypothesis for a pair of replacement methods; that one of the two methods is always greater than the other. The *null hypothesis* H_0 is that no difference can be drawn between the two methods of the pair. When $p < 0.05$, then H_0 is rejected.

Based on the results of Table 7.2, conclusions about the following hypotheses can be drawn:

- H_1**:** The percentage of *generalised entailments* with the popularity replacement function is always greater than the one resulted with the structural and property replacement function. This is also depicted in Figure 8.1.

[2]The t-test is computed in the same way as in the syntactic analysis of regularities (see Chapter 7.2 for details).

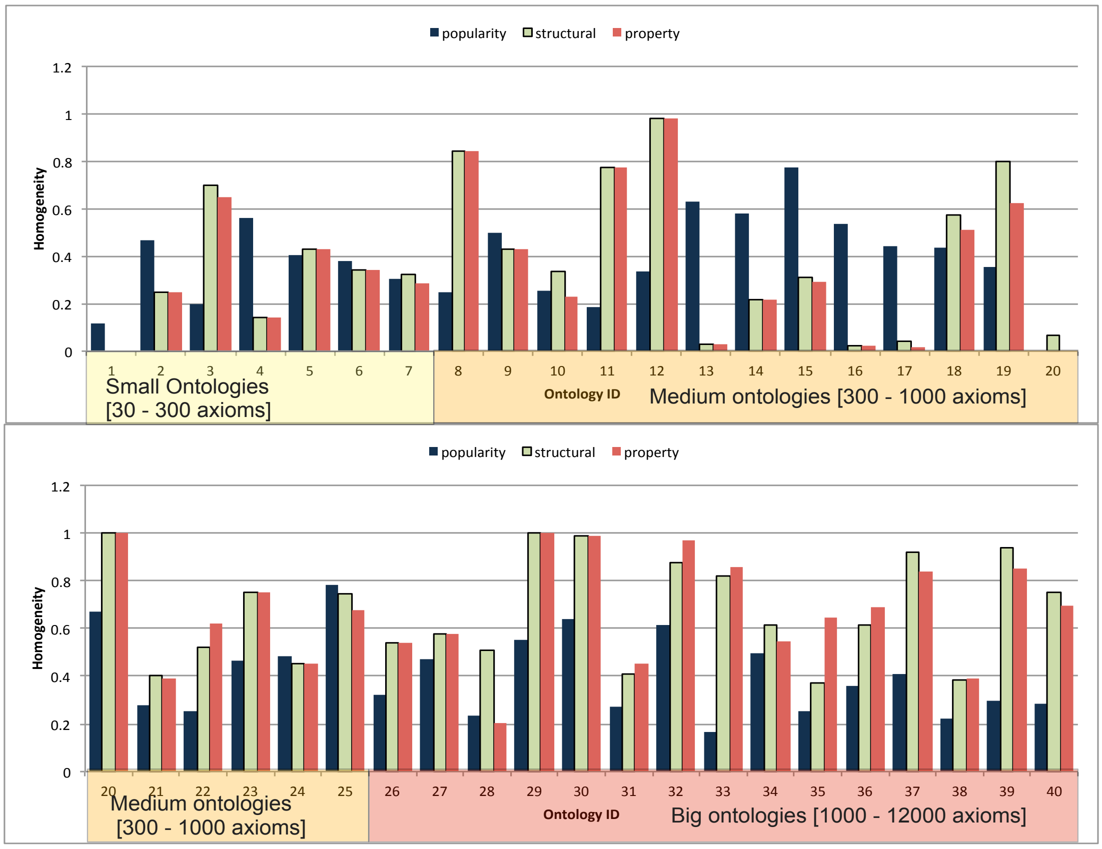


Figure 8.4: Homogeneity of BioPortal Ontologies.

Metric	**Popularity - Property**	**Structural-Property**	**Popularity - Structural**
#Generalised Entailments	$5.28 \cdot 10^{-6}$	$5.15 \cdot 10^{-17}$	8.40 · 10−29
Mean Instantiations per generalisation	0.00016	0.05727	0.00028
Cluster Coverage	0.048	0.457	0.175
AD Compression	8.06 · 10−10	$4.63 \cdot 10^{-3}$	3.03 · 10−9
Homogeneity	9.35 · 10−10	4.80 · 10−1	3.19 · 10−10

Table 8.2: T-Test results for selected metrics. The table shows the p value for checking the significance of difference in regularity metrics between different replacement methods that were used in clustering. Each column represents a potential hypothesis for a pair of methods. The null hypothesis H_0 is rejected for $p < 0.05$.

Thus, in semantic regularity detection, clustering with popularity function finds more regularities.

- H_2**:** Clustering with the popularity replacement function returns always the highest *number of instantiations per generalisation* than the other types of

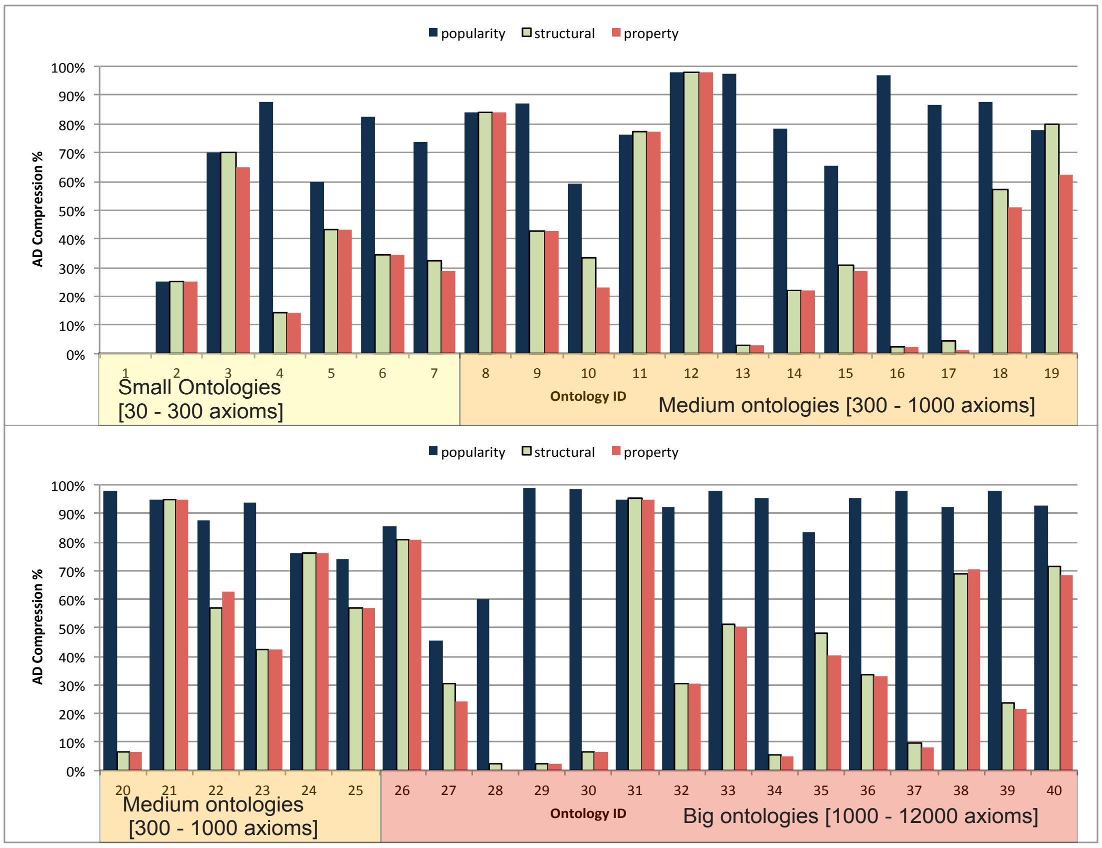


Figure 8.5: AD Compression of BioPortal Ontologies.

clustering. It is nearly valid that the structural replacement function will return generalisations with more instantiations per generalisation than the property replacement function (p=0.057).

- H_3: Clustering with popularity returns regularities with higher *cluster coverage* than clustering with the property replacement function and clustering with the structural replacement function while no conclusion can be drawn on this metric between the pairs of structural and property replacement functions, popularity and structural methods.

- H_4: Clustering with the popularity replacement function results in generalisation that cause the highest *AD compression* compared to the other two methods. Similarly, clustering with the structural replacement function results in generalisations causing higher compression in AD graph than the clustering with the property function.

- H_5: Clustering with property replacement function returns clusters with the highest homogeneity. In addition, clustering with popularity returns the least homogeneous clusters.

Distribution of evaluation metric values. A descriptive summary of the dataset on selected metrics is shown in Figures 8.6–8.10. The boxplots of these figures, can give a better intuition on the distribution of values on regularity for each metric and type of replacement function.

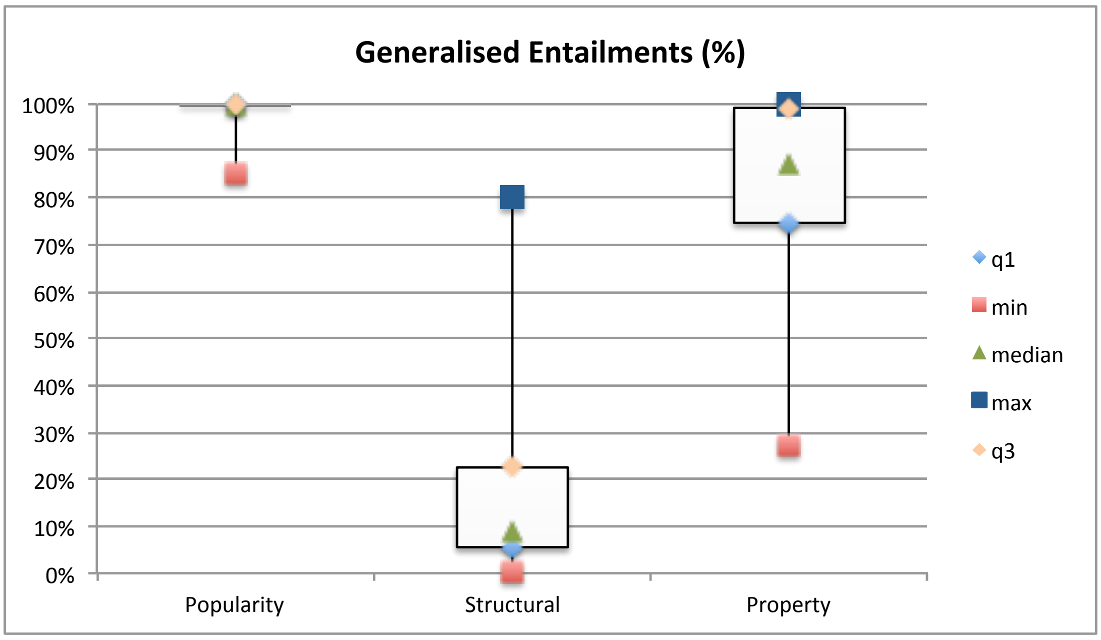


Figure 8.6: Boxplot on generalised entailments.

The general trend for the results obtained for the semantic regularities deviates from the one for the computation of the syntactic ones. As this is depicted in the boxplot of Figure 8.6, all clustering methods have different shape of results. This significance is also verified by the t-test in Hypothesis $H1$.

As shown in Figure 8.7 and also statistically verified in H_2, clustering with the property replacement gives the highest number of instantiations per generalisation, then follows the structural replacement function and last is the property replacement functions. However, as also shown in Figure 8.7 the distribution of values for the structural replacement function is similar to the distribution of values of the property replacement function. For example, in ontology 30 (biological-imaging-methods) in Figure 8.2, the abstraction achieved by the clustering with popularity is the highest (442 entailments per generalisation). But

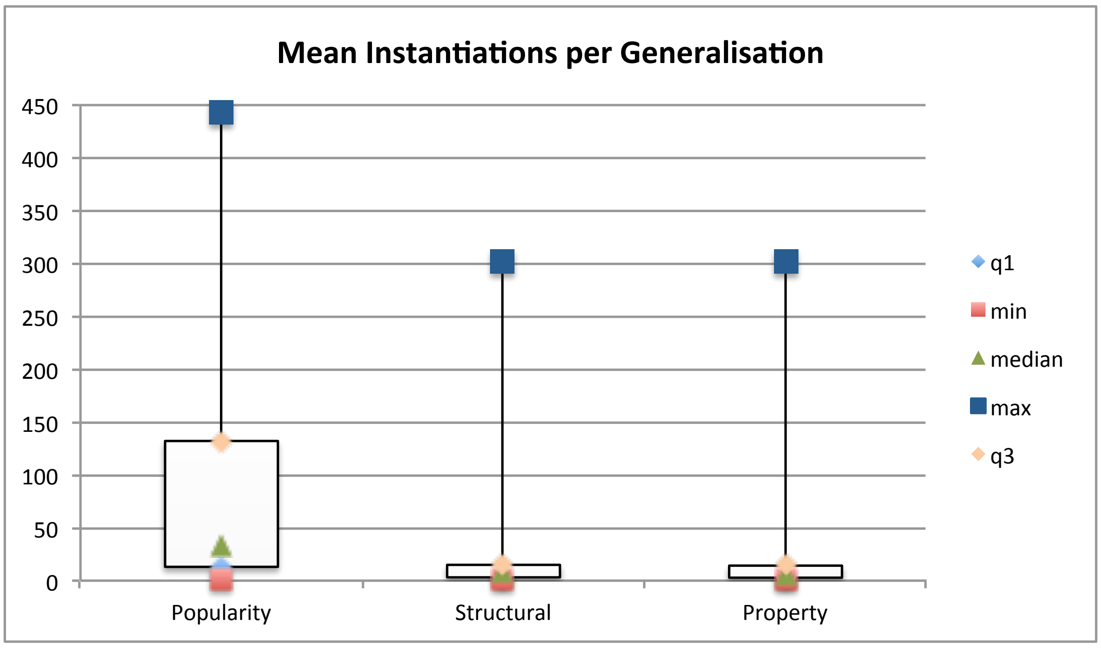


Figure 8.7: BoxPlot on Mean Instantiation per Generalisation of the BioPortal Ontologies.

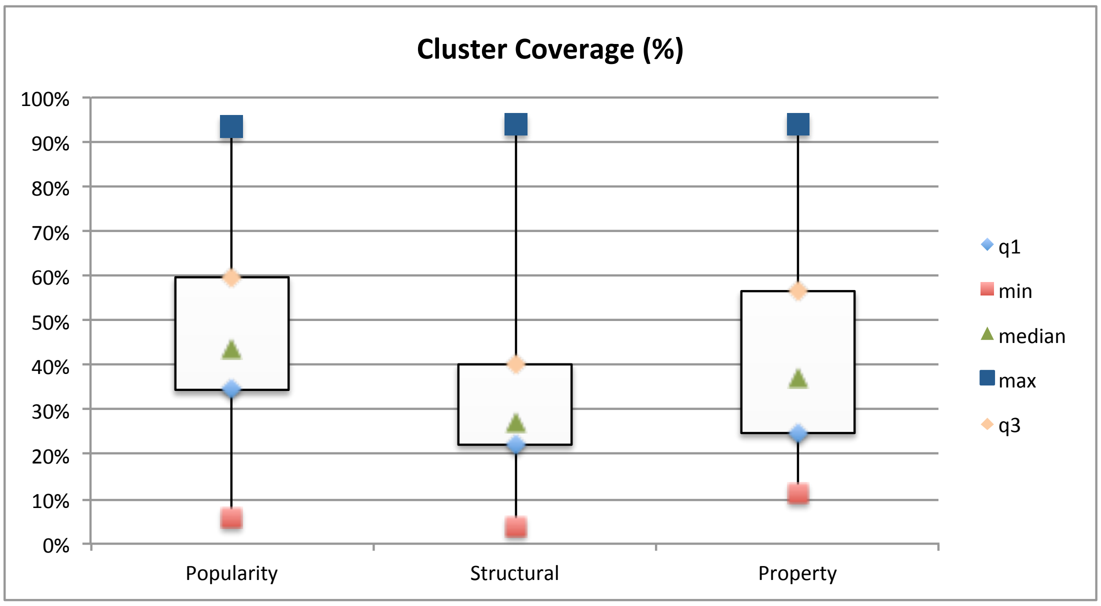


Figure 8.8: Boxplot on Cluster Coverage of BioPortal Ontologies.

the abstraction achieved by the structural replacement function and the property replacement function is the same (1.7 entailments per generalisation).

The reason for this is that the structural and property replacement functions

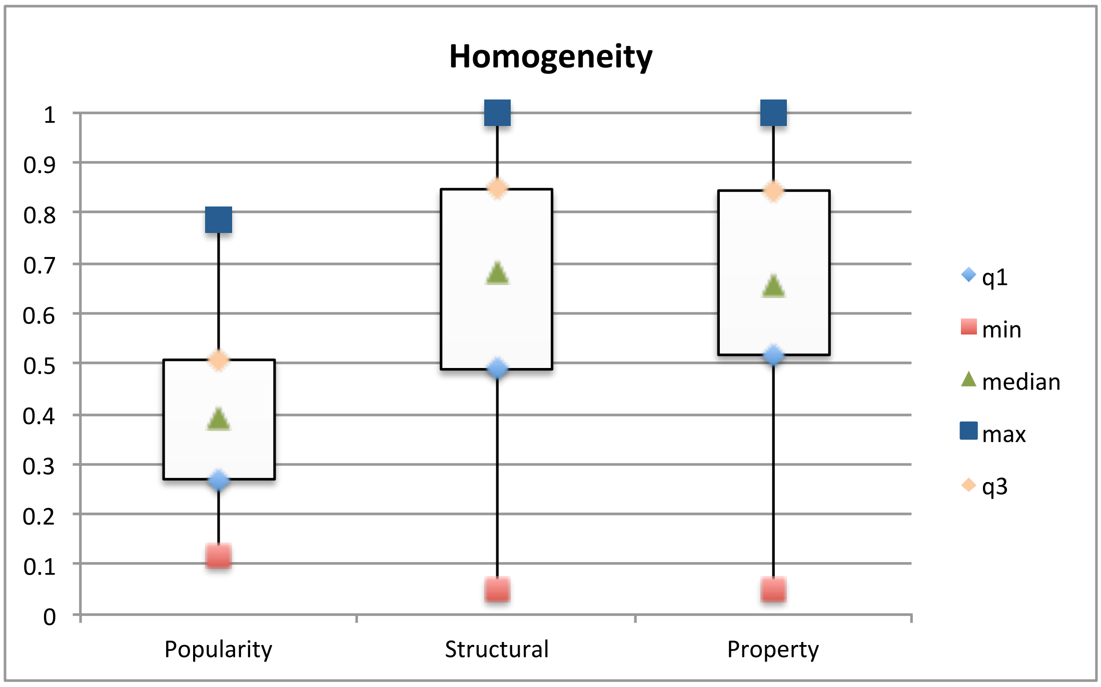


Figure 8.9: Boxplot on the homogeneity of BioPortal ontologies.

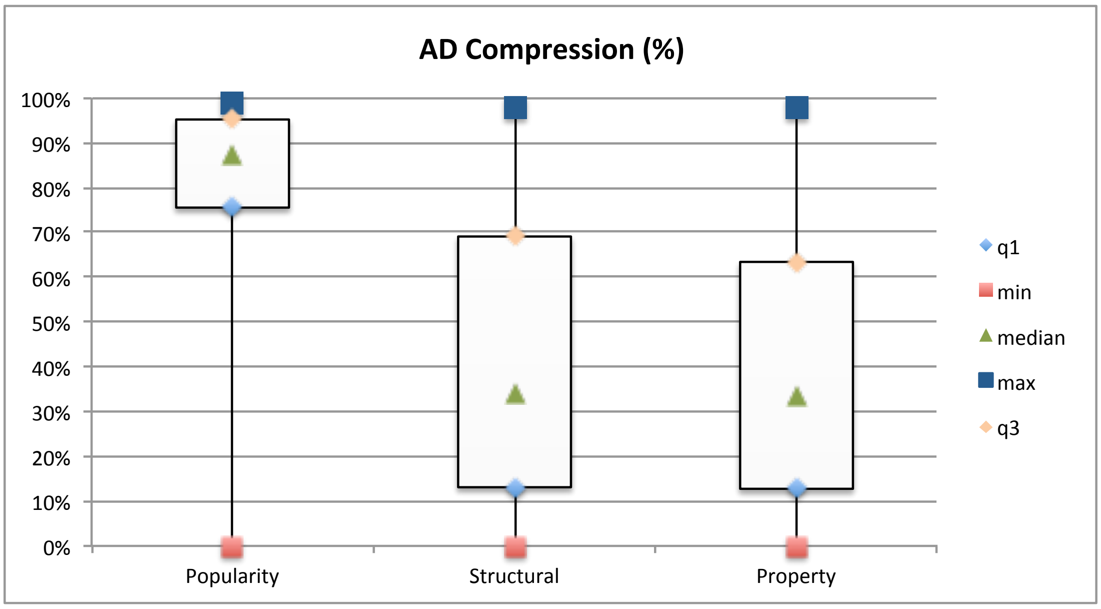


Figure 8.10: Boxplot on AD Compression of the BioPortal Ontologies.

provide a sensitive distance measure, while the popularity replacement function results in a more tolerant replacement function. For ontology 30, clustering with popularity function detected 3 clusters, with 172.3 entities per cluster while clustering with the structural method detected 2 clusters with 23.5 entities per cluster. Thus, entities which belong to a cluster in popularity clustering, are left out in structural clustering ending up in singleton clusters which are not included in the final results.

An example is shown in Figure 8.11. The entailments referring to permeabilized tissue instantiate a regularity for both clustering methods. However the one detected by the structural method is more fine grained; the generalisation has only 2 instantiations while the generalisation for the popularity method instantiates 1424 entailments, amongst them the ones referring to permeabilized tissue.

Generalisation detected in clustering with structural method:
permeabilized tissue *SubClassOf* ?cluster_2
Instantiations: (2)
permeabilized tissue *SubClassOf* Biological Imaging Method
permeabilized tissue *SubClassOf* sample preparation method
where,
{?cluster_2:CLASS=[Biological Imaging Method, sample preparation method]}

Generalisation detected in clustering with popularity method:
?Biological_Imaging_Method *SubClassOf* ?cluster_1
Instantiations: (1424)
Example instantiations:
permeabilized tissue *SubClassOf* Biological Imaging Method
permeabilized tissue *SubClassOf* sample preparation method
where,
{?cluster_1:CLASS=[Biological Imaging Method, sample preparation method]}

Figure 8.11: Example showing how entailments instantiate generalisations of different granularity when a different replacement function is selected in clustering.

Cluster coverage. Similarly to the other metrics, clustering with popularity gives the highest results. This is Hypothesis H_3 which is verified in the t-Test. Also the boxplots in Figure 8.8 depict such difference. In addition, the distribution of cluster coverage values for the property function is more widespread than the other two methods whose distribution is closer to the median value. This is the reason why no clear conclusion can be drawn between the comparison of structural and property function. However, for all of the methods, the 75% of the ontologies had generalisations with more than 20% mean cluster coverage, which is an acceptable metric for assessing the goodness of the generalisations.

Projection of generalisations in the AD. As it have been discussed in Chapter 6, the AD shows that the detected regularities have also an impact in the structure of the AD graph when they are projected in it. It is an external verification that these generalisations are not random, as they also capture existing repetitive structures in the ontology.

The difference of methods on the level of abstraction of the regularities has also an impact on the AD compression. As presented in Figure 8.10, the generalisations that result from the structural clustering cause the highest compression compression of the AD (the 75% of the values are above 70%) while generalisations that result from the structural and popularity clustering cause far less compression (50% of values are above 40% for both methods). This observation is also verified in H_4.

Popularity replacement method. Only three ontologies have AD compression lower than 60%, and of these only one has an AD compression of 0%. The number of clusters in that case is 1, thus it has no impact on the AD of the entailments. The AD compression results are worse for the structural method and popularity, thus it is an indication that these methods might not capture existing regularities in the best way compared to the popularity.

Based on the analysis of the results of this experiment clustering with the popularity method will give generalisations of better quality; more regularities will be detected, with higher abstraction impact over the entailments of the ontology and higher cluster coverage. Thus, the analysis in the following section is based on these results.

Regularity and uniformity of ontologies

All 40 ontologies had some form of regularity in their entailments; in terms of uniformity of regularities, semantic regularities seem to have fewer deviations than syntactic regularities. Figure 8.12 depicts an intuition of the uniformity of the semantic regularities detected by the clustering algorithm with the popularity replacement function. Table 8.3 shows the five ontologies in highest ranking of uniformity. Their ranking is based on the combination of values in Figure 8.12. As shown in the table, the mean cluster coverage of their generalisations exceeds 20%

and these generalisations have a significant abstraction impact on the entailments as the mean instantiations per generalisation is more than 90 entailments.

Ontology/ Axioms	#Clusters	Entities per Cluster	Cluster Coverage(%)	Instantiations per Generalisation	Homogeneity
12 / 467	2	105	76%	302	0.34
13 / 511	3	78	64%	99.1	0.63
16 / 550	3	79.7	58%	117.4	0.54
21 / 708	2	154.5	35%	135.1	0.28
31 / 1 320	3	184.3	28%	204.3	0.27

Table 8.3: Regular and uniform ontologies from the BioPortal corpus. The results for the popularity method are shown.

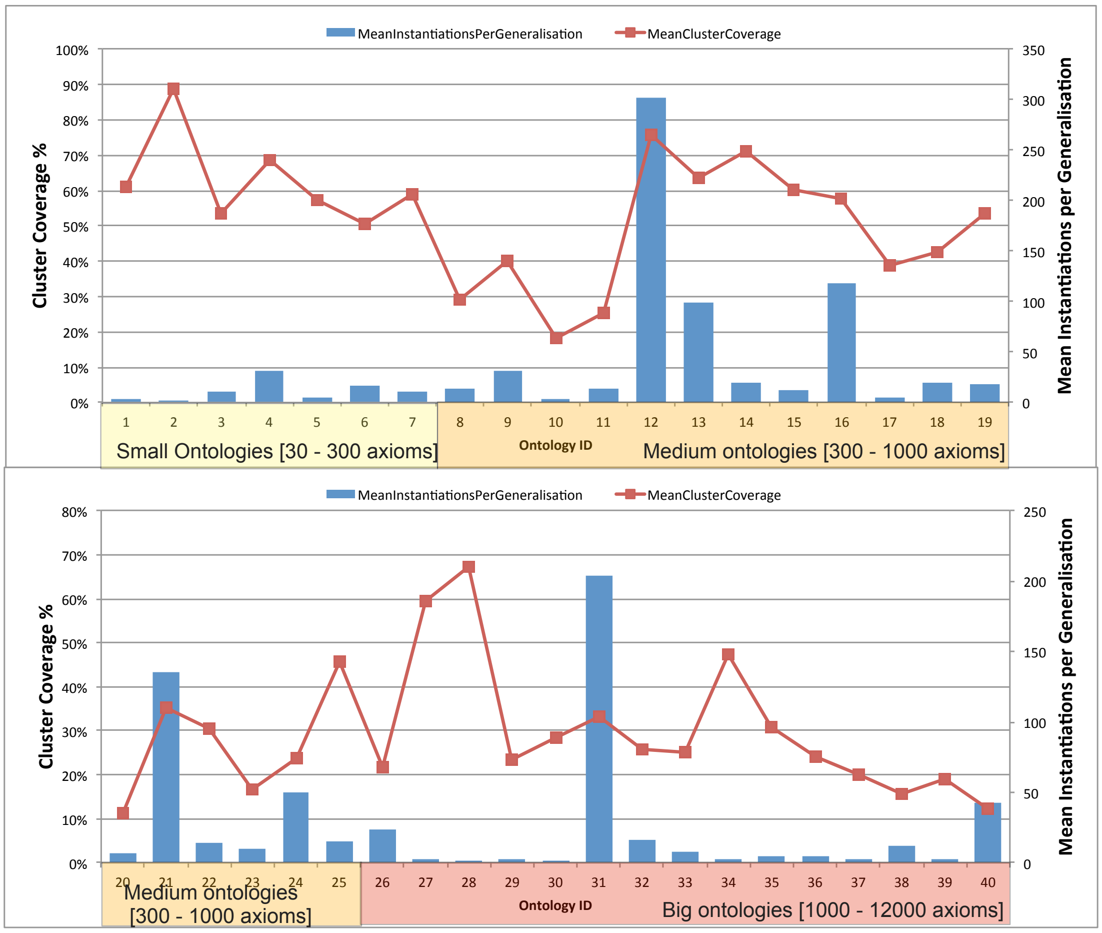


Figure 8.12: Uniformity indication. Cluster coverage and mean instantiations per generalisations for the popularity replacement method

Examples of uniform ontologies. A variety of different types of semantic regularities is captured. The majority of the processed ontologies have very few clusters and few generalisations which abstract a significant amount of entailments.

Ontology 31 (physico-chemical-process) ends up with two clusters of similar entities with clustering using the structural replacement function and with of three clusters when performing clustering with the popularity and property replacement function. The clustering algorithm with popularity method detected 23 generalisations abstracting 4 698 entailments. The main type of patterns of entailments are shown in Figure 8.13. In this ontology, the structural replacement function will achieve a higher abstraction (more instantiations per generalisation) of the ontology. However, the same type of regularities will be highlighted. The remaining 18 generalisations in the ontology are similar to generalisation (4) of Figure 8.13.

Generalisations:
(1) ?cluster_1 *SubClassOf* ?cluster_1
  Instantiations: (3285)
  Example Instantiation:
  simple diffusion *SubClassOf* membrane process
(2) ?cluster_1 *SubClassOf* Part of **some** Thing
  Instantiations: (528)
  Example Instantiation:
  simple diffusion *SubClassOf* Part of some Thing
(3) ?cluster_1 *SubClassOf* Part of some ?physico-chemical_process_ontology
  Instantiations: (464)
  Example Instantiation:
  simple diffusion *SubClassOf* Part of **some** physico-chemical process ontology
(4) ?cluster_1 *SubClassOf* is reverse of **some** ((?cluster_1_conjunction)
  and (Part of **some** ((?cluster_1_conjunction)
  and (Part of **some** ?physico-chemical_process_ontology))))
Instantiations: (66)
Example instantiation:
atomic fluorescence *SubClassOf* is reverse of **some** ((process
and excitation **and** photoexcitation **and** microscopic process)
and (Part of **some** ((process **and** macroscopic process
and absorption **and** photoabsorption)
and (Part of **some** physico-chemical process ontology))))
where,
{?cluster_1:CLASS=[simple diffusion, membrane process, simple diffusion,
  atomic fluorescence],
 ?cluster_1_conjunction:CLASS=[(process **and** excitation
  and photoexcitation **and** microscopic process),
  (process **and** macroscopic process **and** absorption
  and photoabsorption)],
 ?physico-chemical_process_ontology:CLASS=[physico-chemical process ontology]}

Figure 8.13: Example uniform semantic regularities.

Atom Patterns. Similarly, the inspection of the atoms in the AD of entailments that were merged after the projection of the generalisations on the AD shows repetitive structures that occur across different atoms. Such an example is shown in Figure 8.14 taken from ontology 31. The pattern described by two generalisations causes the merging of 358 atoms. In ontology 31, 17 atom patterns were detected.

```
Atom Pattern:
  ?cluster_1 SubClassOf ?cluster_1
  ?cluster_1 SubClassOf 'Part of' some ?'physico-chemical process ontology'
Merged Atoms: (358)
Atom 1: [ATP-hydrolysing chelatase reaction SubClassOf process,
         ATP-hydrolysing chelatase reaction SubClassOf chemical reaction,
         ATP-hydrolysing chelatase reaction SubClassOf biochemical reaction,
         ATP-hydrolysing chelatase reaction SubClassOf phosphorus-containing
                                      anhydride hydrolase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf hydrolase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf transferase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf chelatase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf nucleoside triphosphate
                                      (di)phosphohydrolase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf biotransformation reaction,
         ATP-hydrolysing chelatase reaction SubClassOf macroscopic process,
         ATP-hydrolysing chelatase reaction SubClassOf ligase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf
                                      acid anhydride hydrolase reaction,
         ATP-hydrolysing chelatase reaction SubClassOf 'Part of' some Thing,
         ATP-hydrolysing chelatase reaction SubClassOf 'Part of'
                                 some physico-chemical process ontology]
Atom 2: [bimolecular nucleophilic substitution SubClassOf polar reaction,
         bimolecular nucleophilic substitution SubClassOf
                                      heterolytic substitution reaction,
         bimolecular nucleophilic substitution SubClassOf chemical reaction,
         bimolecular nucleophilic substitution SubClassOf bimolecular reaction,
         bimolecular nucleophilic substitution SubClassOf elementary reaction,
         bimolecular nucleophilic substitution SubClassOf macroscopic process,
         bimolecular nucleophilic substitution SubClassOf process,
         bimolecular nucleophilic substitution SubClassOf substitution reaction,
         bimolecular nucleophilic substitution SubClassOf
                                      nucleophilic substitution reaction,
         bimolecular nucleophilic substitution SubClassOf 'Part of' some Thing,
         bimolecular nucleophilic substitution SubClassOf 'Part of' some
                                      physico-chemical process ontology]
```

Figure 8.14: Example semantic patterns found across many atoms in AD.

8.2 Conclusions on Experiment 4

The empirical analysis of the semantic regularities found in 40 ontologies from BioPortal leads to the following main outcomes:

- The detection of semantic regularities referring to atomic subsumptions and subsumptions between complex class expressions in class level was practical for the 40 ontologies from BioPortal repository. In all of the ontologies some kind of semantic regularity was detected. The number of detected generalisations was between 2 and 1370 generalisations and a total average of 44.3 generalisations per ontology[3]. This indicates that entailments in the ontology can have different structure and can instantiate a pattern. By comparing the results with the syntactic regularities, we observe that the detected semantic regularities are fewer and more uniform; not many deviations are detected as with the syntactic regularities.

- The proportion of entailments that was extracted from the KE is not always analogous to the number of axioms in the ontology. However, the number of entailments is always greater than the number of logical axioms in the ontology. That is because we did not exclude trivial entailments from the data set; that is entailments that are also asserted in the axioms of the ontology. An analysis of the entailments of BioPortal is described in [HPS11]. This analysis is done from a different point of view; it inspects the justificatory structure of the BioPortal ontologies. However, it is remarked that a big number of non-trivial entailments were detected in most of the ontologies. It should be also noted that this analysis was considering only atomic subsumption entailments. To the best of our knowledge, this thesis is the first effort on performing an analysis of entailments including complex class expressions.

- The number and abstraction level of generalisations can differ significantly when a different replacement function is considered in clustering. The empirical results of the semantic regularities of BioPortal showed that the results were more similar for the structural and property replacement function, while popularity gave much higher values for the majority of metrics.

[3]these results refer to clustering with the popularity replacement function

Clustering with the structural and property replacement functions will result in fewer clusters and fewer and more fine-grained generalisations. On the other hand, the popularity replacement function results in more coarse grained generalisations, which instantiate a high number of entailments from the ontology. In addition, the cluster coverage of these generalisations is significantly higher than in the other two methods. This is depicted in Figure 8.8 and verified with the t-Test (Hypothesis H_3). On average cluster coverage is 48% and 75% of the ontologies have more 40% cluster coverage. This shows that the generalisations resulted from clustering with the popularity function are well-formed. Although, clustering with the structural replacement function gave better results for the syntactic regularities, this result is not followed by the analysis for the semantic regularities. A reason for this is that structural replacement function is affected by the length of an entailment (how structural replacement works is explained in Chapter 4.4.3) as the split of a variable entity depends on the other variables in an entailment. The current implementation for extracting entailments from the KE will return entailments whose right part of the entailment consist of long conjuctions of class expressions (e.g. A *SubClassOf* B **and** C **and** D . . .). Structural replacement function is not as effective as the popularity for handing replacements in these long entailments. However, trying breaking down long conjuctions will have as a result a deterioration in the performance of clustering. That is because the number of entailments significantly increases, thus it takes longer to compute the proximity matrix.

- The AD compression results are related to the abstraction impact of the generalisations for most of the ontologies. By computing Pearson's Correlation, a medium positive correlation (0.55) was detected between the AD compression and the mean instantiations per generalisations for the popularity and structural method while a strong correlation (0.6) was detected for the property replacement function. An interpretation is that the input is more homogeneous in terms of regularities, thus the number of instantiations is a good representative for checking whether a good abstraction of the entailments by the generalisations has been achieved. However, all methods will detect highly regular entailments. It remains as a future direction to investigate when it is more useful to use the popularity replacement method for obtaining generalisations with higher abstraction level and when to use structural and property replacement functions for obtaining more fine-grained generalisations. Also, it remains as future direction the definition of better metrics for evaluating how meaningful, with respect to the content of the ontology, the generalisations are. This cannot be claimed through AD yet.

- In terms of uniformity of entailments, on the detection of semantic regularities, the clustering algorithm has a more constrained input; meaning that the entailments whose regularities are computed are only class expressions while for the detection of syntactic regularities a bigger variety of different type of entailments can be considered (e.g. class assertions and property assertions). Thus, by default fewer deviations are expected in the patterns as the input is more constrained for the detection of semantic regularities. Secondly, the semantic regularities show common patterns on the inferences of an ontology. These inferences even though they depend on the asserted axioms, they are not explicitly asserted by the developers of the ontology, but they are generated from the reasoner. Thus, the automatic generation provides also uniformity to the results. In addition, the entailments can be a result of a design pattern used in the asserted axioms of the ontology. Therefore, the generalisations that result are uniform because they depend on the same pattern.

8.3 Experiment 5: Qualitative Analysis of Semantic Regularities

In Experiment 4 of Section 8.1 we demonstrated the ability of RIO to detect semantic regularities in a bigger collection of ontologies from BioPortal. This experiment is showing a more in depth analysis of semantic regularities of a smaller set of ontologies. It should be mentioned that these ontologies were also used in the experiments shown in Sections 7.3 and 7.4 and for analysing their syntactic regularities with respect to expected patterns. This analysis attempts to show differences in the semantic results obtained compared to the syntactic ones.

8.3.1 Experiment Setup

Selected ontologies The selected ontologies for this experiment consist of the SNOMED-CT modules used in Section 7.4 as well as a subset of the ontologies that were used in the Experiment presented in Section 7.3. From these ontologies, OBI is missing as the computation of entailments for this ontology using the KE was impractical.

The setup is similar to the previous experiments and has been described in Section 7.1. In this experiment however we do a very brief comparison of the results and their validation between different clustering methods, as an extensive analysis of the semantic regularities has been performed with a collection of ontologies from BioPortal. We focus mainly on the analysis of regularities and see how they resemble expected patterns. In addition, the same analysis performed for SNOMED-CT's syntactic regularities is repeated considering now only the semantic regularities. We make a comparison with the analysis analysis and discuss about potential irregularities.

8.3.2 Results

Figures 8.15 and 8.16 show the mean instantiations per generalisation and the AD compression respectively while Figure 8.17 shows the mean cluster coverage of the detected semantic regularities. The percentage of entailments that were found to instantiate a regularity in all ontologies is 100%, apart from Amino Acid, whose percentage was 82% only for clustering with the property replacement function.

8.3.3 Discussion

Comparison of the three replacement methods. For this collection of ontologies, the popularity method has a greater abstraction impact on the entailments of the SNOMED-CT-chronic module than the other two methods (shown in Figure 8.15). The structural replacement method has slightly higher number of instantiations per generalisation for the other ontologies. The AD compression shown in 8.16 is more than 50% for all of the methods and ontologies apart from the property replacement method for the Flowers and KUPKB ontology. In that aspect, the property replacement function gives more clusters and more fine grained generalisations but these have higher cluster coverage.

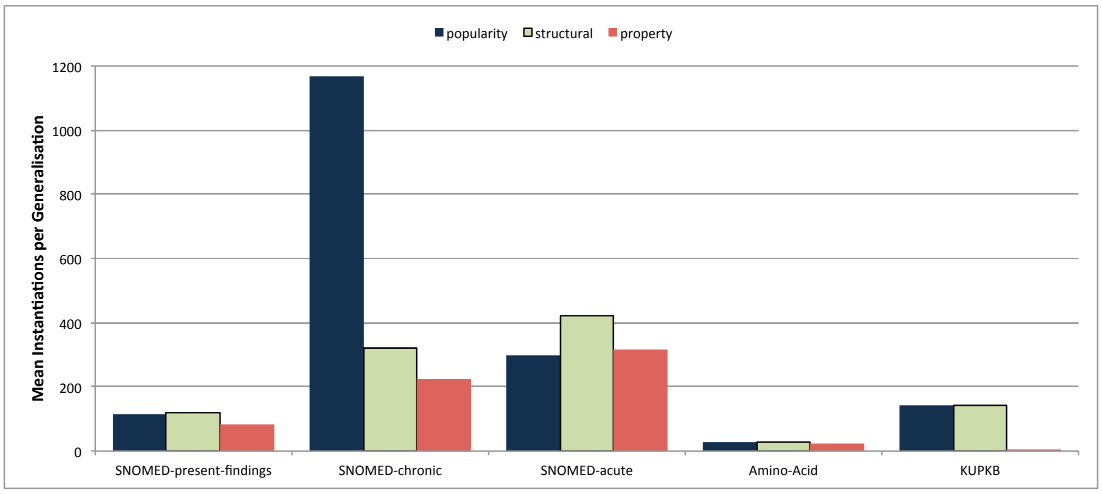


Figure 8.15: Mean Instantiations per Generalisation

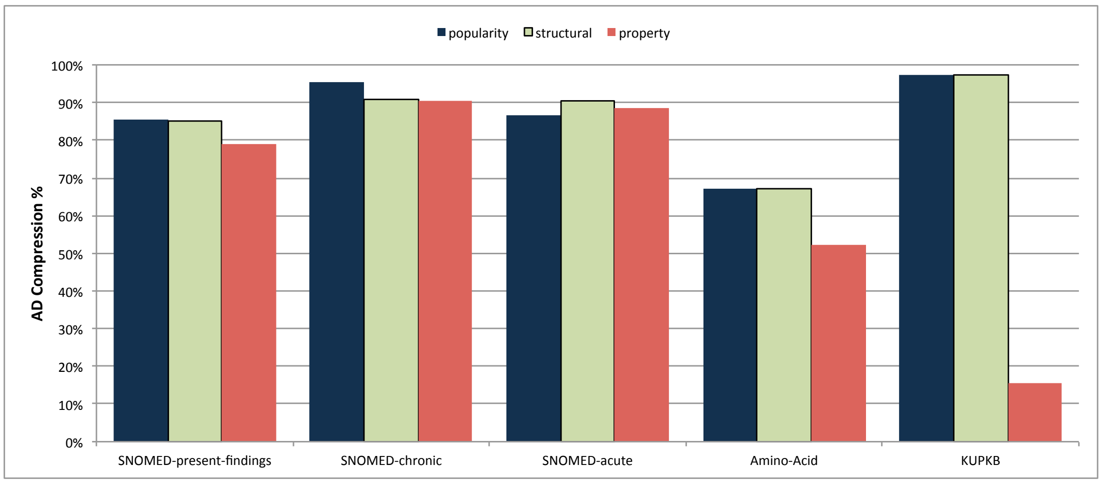


Figure 8.16: Atomic decomposition (AD) compression

Inspection of Regularities

In this section, we will highlight some interesting cases from the inspection of semantic regularities in the ontologies. The examples we illustrate are taken either from the clustering results with the popularity replacement function or the clustering results with the structural replacement function.

KUPKB: The analysis of syntactic regularities of KUPKB, presented in Section 7.3, revealed 10 generalisations referring to two patterns described in the documentation of the ontology [JHI$^+$10] and also shown in Figure 7.23. These

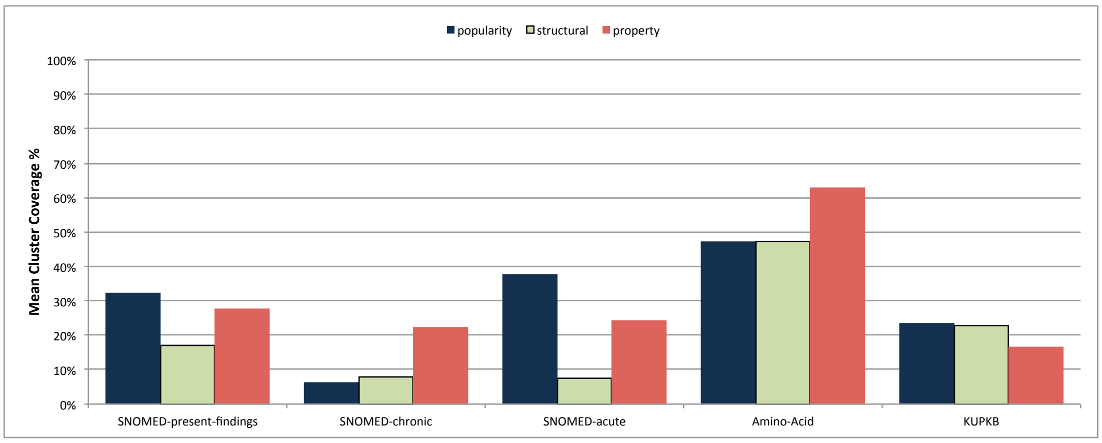


Figure 8.17: Cluster Coverage

two patterns describe kidney cells and they use the properties part_of and participates_in respectively. In the semantic analysis these patterns are covered in a much more uniform way; in particular, each pattern matches a single regularity. The two semantic regularities are shown in Figure 8.18 covering 696 and 360 entailments respectively.

SNOMED-CT modules: Experiment 3, in Section 7.4, showed an extensive analysis of regularities and irregularities with respect to expected patterns in the ontology. However, this analysis was done by considering only the asserted axioms in the ontology. The expected patterns for the SNOMED-CT modules that were used in these experiments were described in 7.4. These are four patterns describing entities which have the keywords “chronic”, “acute”, “present” and “absent” in their label respectively. We will refer to them as *target entities*.

An analysis, similar to the one performed for the syntactic regularities of the four SNOMED-CT modules, is now applied to the semantic regularities of the ontology. As in the analysis of the syntactic regularities for these modules, we will narrow down the inspection and demonstration of results only for regularities and entailments referencing the target entities. Tables 8.4 and 8.5 show the corresponding results from the analysis of the semantic regularities. The following examples and analysis of the results refer to the detection of semantic regularities using clustering with the structural function.

As it is shown in the tables, the analysis of semantic regularities will give a better overview of the regularities and irregularities because the results do not have any syntactic dependencies; the input of the clustering algorithm are

Pattern 1
Generalisation:
?cluster_1 *SubClassOf* part_of **some** (?cluster_2_conjunction)
Total Instantiations: 696
Example Instantiation:
'kidney cortex artery cell; *SubClassOf* part_of
some (cell **and** 'KUPO core entity' **and** 'KUPO anatomy entity')

where
{?cluster_1:CLASS=[kidney cortex artery cell],
?cluster_2_conjunction:CLASS=[
cell **and** 'KUPO core entity' **and** 'KUPO anatomy entity']}

Pattern 2
Generalisation:
?cluster_1 *SubClassOf* participates_in **some** ?cluster_3
Total Instantiations: 360
Example Instantiation:
'kidney proximal convoluted tubule epithelial cell' *SubClassOf*
participates˙in **some** 'bicarbonate transport'
where
{?cluster_1:CLASS=['kidney proximal convoluted tubule epithelial cell'],
?cluster_3:CLASS=['bicarbonate transport']}

Figure 8.18: Semantic regularities example resembling the expected pattern in KUPKB.

	Present	**Absent**
Total number of entities starting with "Present" or "Absent"	0	1
Total number of entities having "present" or "absent" in the middle of their name	59	24
Number of clusters that include the target entities	2	3
Number of generalisations describing the target entities	61	40
Number of instantiations referring to the target entities	832	409
Number of target entities that were not in any cluster	1	0
Number of generalisations which instantiations explicitly refer to the present (Known present (qualifier value)) or absent qualifier (Known absent (qualifier value))	2 (3%)	2 (5%)
Number of instantiations that explicitly refer to the present or absent qualifier	134 (16%)	81 (19%)

Table 8.4: Selected results of the analysis of semantic regularities in present and absent cases.

entailments that have been generated by the reasoner and not been asserted manually by the developer.

Uniformity of regularities. The automatic generation of entailments has as a result the generation of more uniform regularities that do not suffer from syntactic deviations. On the contrary, in the syntactic analysis of regularities, we have seen entities that have similar semantics end up in different clusters because of the different structure of their asserted axioms (for details see Section 7.4). This was reflected both on the number of generalisations capturing the expected

	Chronic	**Acute**
Total number of entities starting with "Chronic" or "Acute"	388	472
Total number of entities having "chronic" or "acute" in the middle of their name	32	38
Number of clusters that include the target entities	2	2
Number of generalisations describing the target entities	96	98
Number of instantiations referring to the target entities	8530	10079
Number of target entities that were not in any cluster	1	1
Number of generalisations which instantiations explicitly refer to the chronic (Chronic (qualifier value)) or acute qualifier (Sudden onset AND/OR short duration (qualifier value))	4 (4%)	2 (4%)
Number of instantiations that explicitly refer to the chronic or acute qualifier	430 (5%)	361 (4%)

Table 8.5: Selected results on the analysis of semantic regularities in chronic and acute cases.

patterns, the number of instantiations per generalisation and the number of entities which were not included in any cluster.

Figure 8.19 shows such a uniform regularity that captures the expected patterns for the "chronic" entities. In particular, the generalisation of the example instantiates 415 entailments which instantiate the expected pattern. Similarly, Figures 8.20, 8.21 and 8.22, show the corresponding semantic regularities capturing the expected patterns for the "acute", "present" and "absent" entities respectively.

Inspection of irregularities

Amino Acid. The generalisation of Figure 8.23 in the AminoAcid ontology revealed that some entities acting as categories for the amino acids like the TinyAliphaticAminoAcid and HydrophobicAminoAcid have a Neutral charge even though the asserted axioms describing these entities are:

Non-PolarAminoAcid *EquivalentTo* AminoAcid **and** (hasPolarity **some** Non-Polar)
HydrophobicAminoAcid *EquivalentTo* AminoAcid **and**
(hasHydrophobicity **some** Hydrophobic)

Thus, they are not expected to be subclasses of the anonymous class hasCharge **some** Neutral.

The reason for this regular entailment can be explained with the justification of Figure 8.24 as it is shown in Protégé 4. Of similar structure is also the justification for the TinyAliphaticAminoAcid. The reason is that the class AminoAcid is defined to be only one of the primitive[4] amino acids (Alanine, Cysteine etc). All of the Amino Acids that can be Hydrophobic have also Neutral charge and the other

[4]Primitive classes are the ones that are not equivalent.

```
Generalisation:
?SCT_138875005 SubClassOf RoleGroup some (Finding site (attribute)
              some (?SCT_138875006_conjunction))
Total Instantiations: 415
Example Instantiation:
'Chronic renal failure syndrome (disorder)' SubClassOf RoleGroup
              some (Clinical course (attribute)
                some (SNOMED CT Concept (SNOMED RT+CTV3)
                and Descriptor (qualifier value)
                and Time patterns (qualifier value)
                and Special atomic mapping values (qualifier value)
                and Courses (qualifier value)
                and Special disorder atoms (qualifier value)
                and Qualifier value (qualifier value)
                and Chronic (qualifier value)))
where
{?SCT˙138875005:CLASS=[Chronic renal failure syndrome (disorder)],
  ?SCT˙138875006:CLASS=[Time patterns (qualifier value),
                        Chronic (qualifier value),
                        Special disorder atoms (qualifier value),
                        Courses (qualifier value),
                        Special atomic mapping values (qualifier value),
                        Qualifier value (qualifier value),
                        Descriptor (qualifier value),
                        SNOMED CT Concept (SNOMED RT+CTV3)]}
```

Figure 8.19: Semantic regularity example resembling an expected pattern in SNOMED-CT for "chronic" entities.

```
Generalisation:
?cluster_1 SubClassOf RoleGroup some (Clinical course (attribute)
             some ?cluster_2)
Total Instantiations: 352
Example Instantiation:
Acute heart failure (disorder) SubClassOf RoleGroup
             some (Clinical course (attribute)
             some Sudden onset AND/OR short duration (qualifier value)) :
where,
{?cluster˙2:CLASS=[Sudden onset AND/OR short duration (qualifier value)],
  ?cluster˙1:CLASS=[Acute heart failure (disorder)]}
```

Figure 8.20: Semantic regularity example resembling an expected pattern in SNOMED-CT for "acute" entities.

Amino Acids are defined to be Hydrophilic. Since hasHydrophobicity property is functional and Hydrophilic is disjoint with Hydrophobic, the amino acids that are

Generalisation:
?cluster_1 *SubClassOf* RoleGroup **some** (Finding context (attribute)
        some (?cluster˙2˙conjunction))
Total Instantiations: 130
Example Instantiation:
Ankle clonus present (situation) *SubClassOf* RoleGroup
some (Finding context (attribute) **some** (Known (qualifier value)
and Finding context value (qualifier value)
and Known present (qualifier value)))
where,
{?cluster_2:CLASS=[Known present (qualifier value), Known (qualifier value),
        Finding context value (qualifier value)],
  ?cluster_1:CLASS=[Ankle clonus present (situation)]}

Figure 8.21: Semantic regularity example resembling an expected pattern in SNOMED-CT for "present" entities.

Generalisation:
?cluster_1 *SubClassOf* RoleGroup **some** (Finding context (attribute)
        some (?cluster_2_conjunction))
Total Instantiations: 79
Example Instantiation:
'Vaginal discharge absent (situation)' *SubClassOf* RoleGroup
 some (Finding context (attribute)
 some (Known (qualifier value)
 and Finding context value (qualifier value)
 and Known absent (qualifier value)))
where,
{?cluster˙2:CLASS=[Known (qualifier value), Known absent (qualifier value),
        Finding context value (qualifier value)],
  ?cluster˙1:CLASS=[Vaginal discharge absent (situation)]}

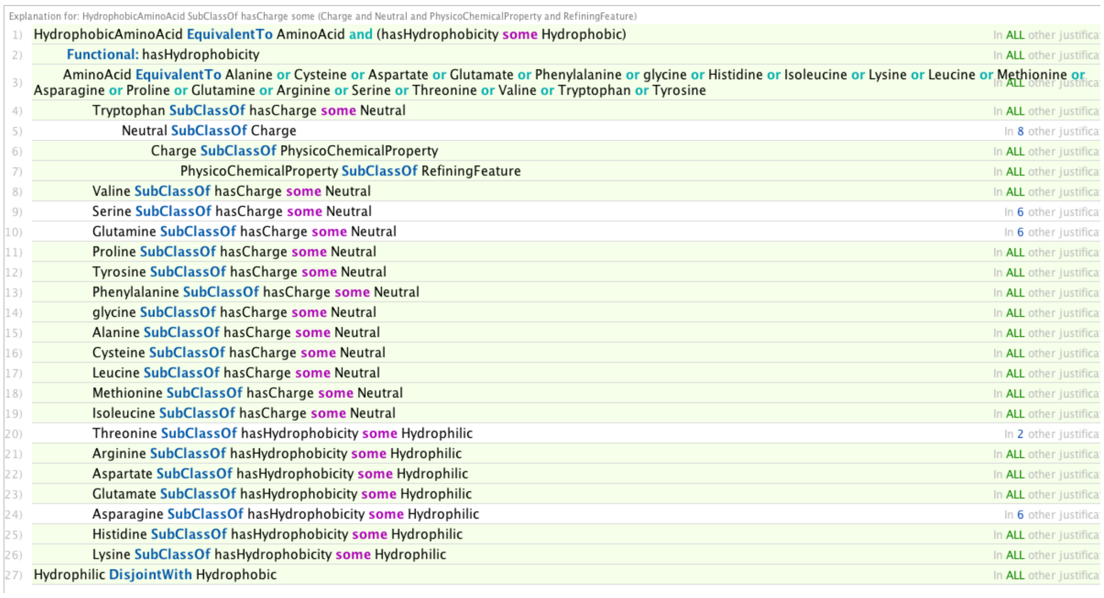
Figure 8.22: Semantic regularity example resembling an expected pattern in SNOMED-CT for "absent" entities.

Hydrophobic must be also neutrally charged. This type of irregularity was not discoverable in the syntactic analysis performed by RIO, as it was not explicitly asserted in the ontology.

SNOMED-CT. In contrast with the analysis of the syntactic regularities, Tables 8.4 and 8.5 show much fewer target classes not included in any cluster. In Section 7.4, Figure 7.34 shows Chronic entities that were not found in any cluster in the syntactic regularities. However, the semantic analysis gave different results.

Generalisation:
?AminoAcid *SubClassOf* hasCharge **some** (?RefiningFeature_conjunction)
Example Instantiations:
TinyAliphaticAminoAcid *SubClassOf* hasCharge **some** (Charge **and** Neutral **and** PhysicoChemicalProperty **and** RefiningFeature)
HydrophobicAminoAcid *SubClassOf* hasCharge **some** (Charge **and** Neutral **and** PhysicoChemicalProperty **and** RefiningFeature)
where,
{?AminoAcid:CLASS=[TinyAliphaticAminoAcid, HydrophobicAminoAcid],
?RefiningFeature_conjunction:CLASS=[(Charge **and** Neutral **and** PhysicoChemicalProperty **and** RefiningFeature)]}

Figure 8.23: An example semantic "irregularity" in the Amino Acid ontology. The TinyAliphaticAminoAcid and the HydrophobicAminoAcid are also neutrally charged, which is not expected as both their axiomatic description and label do not indicate such property.



```
Explanation for: HydrophobicAminoAcid SubClassOf hasCharge some (Charge and Neutral and PhysicoChemicalProperty and RefiningFeature)
 1) HydrophobicAminoAcid EquivalentTo AminoAcid and (hasHydrophobicity some Hydrophobic)          In ALL other justifica
 2)     Functional: hasHydrophobicity                                                              In ALL other justifica
 3)   AminoAcid EquivalentTo Alanine or Cysteine or Aspartate or Glutamate or Phenylalanine or glycine or Histidine or Isoleucine or Lysine or Leucine or Methionine or
    Asparagine or Proline or Glutamine or Arginine or Serine or Threonine or Valine or Tryptophan or Tyrosine   In ALL other justifica
 4)         Tryptophan SubClassOf hasCharge some Neutral                                           In ALL other justifica
 5)             Neutral SubClassOf Charge                                                          In 8 other justifica
 6)                 Charge SubClassOf PhysicoChemicalProperty                                      In ALL other justifica
 7)                     PhysicoChemicalProperty SubClassOf RefiningFeature                         In ALL other justifica
 8)         Valine SubClassOf hasCharge some Neutral                                               In ALL other justifica
 9)         Serine SubClassOf hasCharge some Neutral                                               In 6 other justifica
10)         Glutamine SubClassOf hasCharge some Neutral                                            In 6 other justifica
11)         Proline SubClassOf hasCharge some Neutral                                              In ALL other justifica
12)         Tyrosine SubClassOf hasCharge some Neutral                                             In ALL other justifica
13)         Phenylalanine SubClassOf hasCharge some Neutral                                        In ALL other justifica
14)         glycine SubClassOf hasCharge some Neutral                                              In ALL other justifica
15)         Alanine SubClassOf hasCharge some Neutral                                              In ALL other justifica
16)         Cysteine SubClassOf hasCharge some Neutral                                             In ALL other justifica
17)         Leucine SubClassOf hasCharge some Neutral                                              In ALL other justifica
18)         Methionine SubClassOf hasCharge some Neutral                                           In ALL other justifica
19)         Isoleucine SubClassOf hasCharge some Neutral                                           In ALL other justifica
20)         Threonine SubClassOf hasHydrophobicity some Hydrophilic                                In 2 other justifica
21)         Arginine SubClassOf hasHydrophobicity some Hydrophilic                                 In ALL other justifica
22)         Aspartate SubClassOf hasHydrophobicity some Hydrophilic                                In ALL other justifica
23)         Glutamate SubClassOf hasHydrophobicity some Hydrophilic                                In ALL other justifica
24)         Asparagine SubClassOf hasHydrophobicity some Hydrophilic                               In 6 other justifica
25)         Histidine SubClassOf hasHydrophobicity some Hydrophilic                                In ALL other justifica
26)         Lysine SubClassOf hasHydrophobicity some Hydrophilic                                   In ALL other justifica
27) Hydrophilic DisjointWith Hydrophobic                                                           In ALL other justifica
```

Figure 8.24: Justification for the entailment HydrophobicAminoAcid *SubClassOf* hasCharge **some** (Charge **and** Neutral **and** PhysicoChemicalProperty **and** RefiningFeature)

From the 12 entities of Figure 7.34, entities 'Chronic progressive renal failure (disorder), Chronic inflammatory demyelinating polyneuritis (disorder) instantiate the chronic pattern (see Figure 8.19) in their entailments. These were excluded from the syntactic regularities as they were not instantiating the pattern explicitly.

Similarly, from the 11 acute entities of Figure 7.35 that were not included in any cluster in the syntactic regularities, the Acute cardiac pulmonary edema (disorder) only was found to instantiate the acute pattern (see Figure 8.20).

Lexical irregularities. In addition, the analysis of semantic regularities revealed entities whose referencing entailments were instantiating one of the expected patterns, but this was not followed by the label of their name. An example of such entity is shown in Figure 8.25. Entity Poliomyelitis osteopathy of the forearm (disorder), even though it does not have in its name the keyword "Acute" like the other acute disorders, is inferred to be an Acute disease (disorder). Whether this is an intended irregularity or not is not clear. However, it is useful to have tools for helping in the isolation of such irregularities. In addition, it is more difficult to detect such entities with the analysis of the asserted axioms, since such information is not explicitly asserted in the description of the entity but it propagates through the class hierarchy.

This also justifies why the number of entities with a particular keyword in their labels is lower than the number of entailments that refer to the corresponding pattern. It should be noted that it is expected that a target entity to have only one instantiation referencing the corresponding "qualifier" entity (e.g. 'Chronic (qualifier)', 'Sudden onset AND/OR short duration (qualifier value)', 'Known present (qualifier value)', 'Known absent (qualifier value)'). In particular, there are 388 chronic entities, from which we have shown that 10 of them do not instantiate the pattern, and 430 instantiations of the corresponding pattern. Thus, there are 10 entities that are implied to be chronic diseases but this is not indicated in their label. This might be an intended deviation from the developers or mistake in the labeling of these entities. In the scope of this work is to isolate such discrepancies. Similarly, for the SNOMED-CT-acute module, there are 8 entities, which instantiate the "acute" pattern without having the keyword "acute" in their label. In the other modules, there are 75 such entities, which are implied to be Present clinical findings and 56 entities which are implied to be Absent clinical findings. It should be noted that in these 56 entities, the class No edema present (situation) was included. Thus, a "present" entity was implied to be an Absent clinical finding.

```
?cluster_1 SubClassOf RoleGroup some (Associated with (attribute) some
        ((?cluster_1_conjunction and ?cluster_2_conjunction)
        and (RoleGroup some (Clinical course (attribute)
          some ?cluster_2))))

Poliomyelitis osteopathy of the forearm (disorder) SubClassOf RoleGroup
     some (Associated with (attribute)
     some ((Disease due to Picornaviridae (disorder)
     and Disorder of nervous system (disorder)
     and Acute nervous system disorder (disorder)
     and Acute disease (disorder)
     and Viral disease (disorder)
     and Disorder of body system (disorder)
     and Acute poliomyelitis (disorder)
     and Clinical finding (finding)
     and Infectious disease (disorder)
     and Disease due to Enterovirus (disorder)
     and Acute infectious disease (disorder)
     and Disease (disorder))
     and (RoleGroup some (Clinical course (attribute)
     some Sudden onset AND/OR short duration (qualifier value))))) :
where {?cluster_2:CLASS=[Clinical finding (finding), Disease (disorder),
    Sudden onset AND/OR short duration (qualifier value)],
    ?cluster_1:CLASS=[Acute nervous system disorder (disorder),
    Viral disease (disorder), Acute infectious disease (disorder),
    Poliomyelitis osteopathy of the forearm (disorder),
    Infectious disease (disorder), Disorder of nervous system (disorder),
    Disease due to Picornaviridae (disorder),
    Acute disease (disorder), Disease due to Enterovirus (disorder),
    Disorder of body system (disorder), Acute poliomyelitis (disorder)]}
```

Figure 8.25: Example of an entity (Poliomyelitis osteopathy of the forearm (disorder)) that is inferred to be an Acute disease (disorder) without complying to the corresponding naming convention.

8.4 Conclusions on Experiment 5

From the analysis of the semantic regularities reported above, we can conclude the following:

- The semantic analysis revealed regularities which could not be detected by a syntactic tool. For example, in the amino acid ontology a pattern used in the asserted axioms of the ontology had as a consequence the same type of an entailment captured by the regularities. These regularities also had similar structure in the justifications. This can be a motivation for future work on additional assessment of goodness of the semantic regularities. In particular, entailments of similar structure are expected to have similar justifications and justificatory structure. Measuring such similarity between justifications of entailments instantiating the same generalisation can be an additional metric for assessing the quality of semantic regularities.

- Comparison between the syntactic and semantic regularities of SNOMED-CT showed that the semantic regularities reflecting an expected pattern are more uniform; fewer generalisations were detected with much higher abstraction impact. This can give a better overview of irregularities in the ontology with respect to an expected pattern: the semantic regularities abstract over the semantics of an ontology, thus entities that do not explicitly instantiate a pattern, can be found to instantiate it in the entailments of the ontology. Thus, a better picture can be gained about deviations of a pattern.

- The analysis of semantic analysis revealed entities that were not included in the target entities, meaning not having a particular keyword in their label; yet, they seemed to instantiate the corresponding pattern mapped to a keyword. In addition, this observation could not be made with syntactic regularities or with the OPPL script. That is firstly, because this information was implicit in the axioms of the ontology and secondly, because the label of the target entities was used as a parameter for the corresponding OPPL query (an example of OPPL query is shown in Figure 7.36 for the "Chronic" entities).

- The use of KE will produce entailments of the type $\mathsf{A} \sqsubseteq \mathsf{B}$, $\mathsf{A} \equiv \mathsf{B}$. In these, the right hand side of the entailment (superclass) is usually a long conjunction of class expressions. Algorithm 4, which does the extraction of the entailments, will return such output. As shown in the experiments presented above in this Chapter, this long conjunction can affect the results of the replacement function since various replacements implicitly depend on the length of the axiom. It remains as a future implementation improvement to simplify the entailments and separate conjunctions. At the moment this procedure adds cost to the computation of the clusters as the simplification will lead to the increase of referencing entailments per entity. Thus, this makes the computation of the proximity matrix slower.

- For quality assurance, adding more entailments to the set such as class assertions, property assertions and so on can extend the analysis of RIO in other parts of the ontology and consider a more full set of its semantics. At the moment, this change adds computational cost. However, the ultimate goal is to harmonise modularity with reasoner computation and regularity detection for faster and more systematic ontology quality assurance.
- KE in combination with RIO and OPPL can help towards a more systematic quality assurance of ontologies. RIO can be used for guiding the observation of fragments of the ontology and OPPL for the verification of discrepancies and irregularities in the ontology. Also using the lexical analysis of entities in combination with their description and semantics in the ontology is a good starting point to check if naming conventions comply with axiomatic descriptions. Such lexical pattern guided the analysis with SNOMED-CT.

Chapter 9

Conclusions and Future Work

9.1 Thesis Overview

This thesis deals with the problem of detecting syntactic and semantic regularities in ontologies. We presented RIO, a framework that uses clustering for detecting regularities in the asserted axioms and entailments of an ontology. The regularities are expressed as generalisations, including variables, representing corresponding clusters of similar entities. These generalisations are abstractions over the axioms and entailments of an ontology over the repetitive structures that are found. As demonstrated through a series of experiments in the previous Chapters, regularities can be revealed as patterns in the ontology and these could be used to gain an intuition of the construction of an ontology. In addition, regularities in the entailments of an ontology can give an intuition of the semantics of an ontology; meaning entailments are grouped according to their similarities in the structure, revealing how entities of similar design participate in entailments of similar structure. Also, it was demonstrated that regularities can be useful for guiding the quality assurance of an ontology; for highlighting entities whose design deviates from the expected one. A number of internal and external criteria, mainly including a variety of metrics for analysing the proximity matrix and assessing the goodness of clusters and final generalisations, were used to verify that the results are reliable.

As discussed in Chapter 2, these results can be put to use in reverse engineering an ontology to understand how it works and how it was built, and they help in providing a simpler, more abstract view of the ontology, which highlights possible errors as irregularities, as described in Chapters 7, 8.

9.2 Answers to the Main Research Questions

Each research question has been answered at a different point in the thesis by verifying the related hypothesis introduced in Chapter 1. The combination of these answers shows that the corresponding hypotheses are supported. These can be summarised as follows:

9.2.1 How can syntactic and semantic regularities be detected in ontologies?

This question is related to the first and second hypotheses presented in Chapter 3, Section 3.3: syntactic regularities in an ontology can be expressed with generalised axioms, which use variables to hold similar entities. An example is:

?Vehicle *SubClassOf* hasPart **some** ?VehiclePart

where ?Vehicle, ?VehiclePart are variables holding similar entities. The main challenge in this problem is how to bind the variables, i.e., how to detect the groups of similar entities that are going to be represented by a variable. In the literature, this is categorised as a problem of *unsupervised pattern recognition* [Han07, Lan05, BN06, TK06].

The way we deal with such problem in this thesis is with the use of clustering *clustering*, as it enables meaningful partitioning of data into groups. In the approach taken here, we used hierarchical agglomerative clustering; one of the simplest clustering algorithms for the detection of clusters in data. For the detection of syntactic regularities, a clustering algorithm will detect clusters of similar entities based on their usage in the *asserted axioms* of an ontology. Similarly, for the detection of semantic regularities, the clustering algorithm used in RIO will detect clusters in the *entailments* of an ontology.

9.2.2 How can the validity of the detected regularities be assessed?

Methods for assessing the validity of the detected regularities were presented in Chapter 6. In brief the validity of the detected regularities is assessed using two categories of criteria:

- **Internal criteria:** Internal criteria consider the actual results and data used in the cluster analysis for assessing the quality of the results. A number of metrics have been defined. These are *clustering metrics* for assessing the compactness and separation of the data clusters as well as *generalisation metrics* for assessing the goodness of the detected generalisations, such as their number, their abstraction impact in the ontology, their cluster coverage etc. In addition, these metrics, in combination with selected external criteria, are used for assessing the uniformity of an ontology, i.e. how complex is the ontology with respect to the inspection of its patterns.

- **External criteria:** These consider external structures and parameters which are independent from the methods used in clustering. For external criteria we used the *reference to ontology documentation for developers*, *verification with OPPL scripts*. We also defined a metric based on the projection of the generalisations on the the modular structure of an ontology exposed by the Atomic Decomposition (AD) graph [VPS11]. However, the first two methods are the most reliable methods for verifying the existence of patterns. The investigation of alternative evaluation metrics with respect to the AD remains as a future work.

In Chapters 7 and 8 we presented a series of experiments demonstrating the usage of these criteria for the assessment of the regularities and their effectiveness in measuring the quality of generalisations and clusters; the experiments proved that the detection of regularities is feasible and a useful instrument to detect patterns and for quality assurance.

9.2.3 How can regularities reveal the compositional style of an ontology?

A common technique during the development of an ontology is for the developers to adopt design templates for describing different fragments of the ontology. The work around Ontology Design Patterns verifies this [IRS09, GP09, Gan05, BS05]. In this thesis we have described different notions of patterns and regularity (Chapter 2). By making the assumption that instantiating a pattern in an ontology results in repetitive structures in the axioms of the ontology, the reverse process that detects such repetitive structures can reveal the original patterns. Of course, the detection and expression of these repetitive structures is not as accurate as the initial pattern, but the approximation is good enough to group axioms that have been developed in the same way. RIO performs this task by abstracting axioms with generalisations. It should be noted that a pattern will be exposed by RIO if it has been used more than once in its asserted axioms or if it has a big impact in the semantics of an ontology. Such examples were highlighted in the experiments of Chapters 7, 8; e.g. in SNOMED-CT when an expected pattern described in the documentation of the ontology was not explicitly instantiated by all the entities with the corresponding naming convention (See for example Figures 7.34, 7.35, in Section 7.4). However, the analysis of the semantic regularities showed that for many of these entities the pattern was instantiated in the entailments.

9.2.4 How can semantic regularities reveal queries about an ontology?

Semantic regularities are abstractions over the entailments of an ontology by grouping entailments with similar structure. This type of regularities first contributes to the exposure of the compositional style of an ontology. Secondly, they reveal similarities in terms of semantics of an ontology; how fragments of the ontology work in a similar way and an intuition about what can be queried about these fragments. Until now, the only way to achieve this is by manually writing DL queries. This work provides tools to help this process along, with data extracted from the ontology rather than guessed by the writer.

9.2.5 How can the analysis and inspection of regularities reveal irregularities in an ontology?

Irregularities can be revealed by inspection at the following levels of analysis:

- With regularities of similar structure, usually deviating in only one variable. The deviating variable represents two clusters, which did not get merged even though they have at least one axiom in common (the one represented by the generalisation). The reason is that these clusters had more differences in their remaining referencing axioms than commonalities. This is an indication of a deviating pattern (an example can be found in Section 7.2 in Figure 7.12.

- With clusters not including entities which were expected to instantiate a pattern. Such examples were highlighted in the analysis of the syntactic regularities of SNOMED-CT; where a number of entities were excluded from clusters as they were missing the instantiation of a pattern. These were marked as irregularities.

9.2.6 How can regularities provide a summary of the ontology?

The syntactic regularities are expressed with generalisations abstracting over the asserted axioms and semantic regularities are abstractions of the entailments in an ontology. First of all, the generalisations are fewer than the actual number of axioms and entailments as they are abstractions. Secondly, these abstractions have similar syntax with the low level instantiations.

9.3 Thesis contributions

The main contributions of this thesis are:

1. Methods for the detection of repetitive structures in the syntax of the asserted axioms of an ontology. These are named as syntactic regularities. Their detection is done by using clustering. The final results are generalised axioms, named as generalisations, which are formed with respect to the detected clusters. In particular, they include variables representing a group of similar entities as detected by the clustering. The generalisations are abstractions of the axioms in the ontology; they group axioms that have a similar design.

2. Methods for the detection of repetitive structures in the entailments of an ontology. These are named as semantic regularities. The semantic regularities are also detected in the same way as the syntactic regularities. However, in the semantic approach the input to the clustering phase is abstracted away from the syntax, i.e., the particular choice of axioms to model a concept, but it only depends on the semantics; this removes noise produced by alternative equivalent choices made by the ontology developers (e.g. asserted axioms), providing insight that could not be reached with the syntactic approach.

3. The methods for the detection of syntactic and semantic regularities comprise the RIO framework. The framework has been applied to a wide variety of ontologies to verify its ability to detect regularities.

4. Evaluation methods for assessing the validity and quality of the results. These are standard metrics for assessing the validity of clusters as well as metrics for assessing the quality of regularities; their number, their abstraction level over the axioms and entailments of an ontology, their cluster coverage ability and so on. In addition, a number of external criteria such as analysis of regularities with respect to expected patterns described in the documentation and being verified with OPPL scripts were used for assessing the quality as well giving a picture of how meaningful they can be. Finally, metrics for measuring the impact of change after projection of regularities in independent structures such as the modular structure of an ontology were defined.

5. We made contribution towards the reverse engineering of ontologies: the regularities being found can reveal the construction of an ontology and the adopted patterns by the developers.

6. A contribution towards quality assurance to ontology. As the regularities found enable to abstract away from details, they vastly reduce the complexity of an overview of the ontology. This simplifies the detection of particular types of errors, such as missing information about a term.

9.4 Future Work

This thesis has introduced a novel framework for detecting regularities in ontologies. While the methods described can be used to address practical problems with ontologies, the research has opened paths for future work which are presented below.

9.4.1 Ontology Comprehension

As discussed in Chapter 2, ontology comprehension is the broader field in which this research is placed. Although ontology comprehension is not the goal of this thesis, it has been the initial motivation for the RIO framework. The methods of

RIO are placed in the area of reverse engineering of ontologies, with the goal of integrating existing methods and new ones in an ontology comprehension framework. Pattern based development of ontologies and pattern detection in them is a key aspect in reverse engineering. Pattern detection named as *cliché recognition* has been used in software comprehension frameworks [RS02, vMV97, NDG05]. They are used for highlighting parts of the code that are similar. In ontology engineering, there have been many efforts towards the definition and usage of ODPs. However, even in the absence of ODPs, the development of ontologies with knowledge patterns is a best practice, leading to the development of methods such as the OPPL scripting language and spreadsheet templates for populating an ontology.

As seen in the results of the empirical evaluation, metrics on the goodness of regularities and uniformity of an ontology can give an intuition of how the ontology has been composed, deviations of regularities and so on. Such methods could help the inspection and authoring of ontologies as a part of an ontology comprehension framework.

An additional future task in this direction is the coupling of RIO with ontology versioning methods like the ones described in [GPS11, GPS12] for a pattern based analysis of ontology evolution. In particular it would be interesting to explore how developers update an ontology with respect to its patterns; if they are working continuously with a pattern between different versions of an ontology or if regular design is interrupted, changed or dropped between different versions. All these are promising directions worth investigating.

9.4.2 Ontology Quality Assurance

The analysis of SNOMED-CT syntactic and semantic regularities showed how RIO can be used as part of a systematic quality assurance of ontologies. Improvement of the methods of RIO for more quality assurance is in the immediate future plans of RIO framework. In particular, the analysis that was done manually on the results could be automatised in RIO to filter information and narrow down the scope of regularity inspection on the fly, to lighten the burden on the user once the tool completes its investigations. This could, for example, include the analysis on terms labeling in bioinformatics ontologies like the Foundational Model of Anatomy ontology (FMA), Gene Ontology (GO), and SNOMED-CT, and enriching the concept definitions prior to regularity detection [QMFBS12].

Additional methods that seem worth investigating include the analysis of generalisations of similar structure for isolation of deviation. In addition, towards this direction would be the further analysis of the proximity matrix and the highlight of clusters that are closer to each other in terms of distance. This is an indication of existent similarities but also deviations in the referencing axioms of these clusters of entities. Other methods will include the combination of RIO with AD decomposition; it has been used in this thesis as an external criteria for assessing the goodness of generalisations by measuring the rate of compression of the AD graph after generalisations. A further analysis of this compression is the isolation of atoms which included axiom instantiating a regularity detected RIO but the corresponding atoms did not get merged. These can be potential deviations of patterns in the ontology.

9.4.3 Ontology Integration

Ontology integration is another task in which RIO could be useful. Existing methods for ontology integration mainly focus on metrics used for the analysis of an ontology. The authors in [ŠZSI10] refer to a pattern based approach towards ontology alignment. However, in the approach used in [ŠZSI10] the user has to manually express the pattern with OPPL for finding common fragments between the ontologies for integration. RIO could facilitate such a procedure by exposing the regularities of the ontologies which can be used as a starting point for the alignment.

9.4.4 Pattern Induction

The regularities detected by RIO highlight potential patterns in the ontology. However, not all regularities necessarily represent a well defined pattern resembling the construction of an ontology. There can be regularities but these are not necessarily mapped to a pattern. In addition, combination of generalisations can constitute a pattern that fully describes a set of entities and this can be missed in the current version of RIO. RIO's regularities can be one level below the patterns in an ontology, as they are a statistical approach, thus the final generalisations do not capture patterns as they were initially designed by ontology developers. Additional filtering can be required in cases where many generalisations are detected; RIO's clustering technique facilitate such procedure, as the generalisations that are built are expressed with respect to the detected clusters in the ontology. Thus, the set of generalisations referencing a variable describe also the corresponding cluster. However, the full description of the terms in a cluster with a single generalisation is not always feasible. In particular, the variable of a specific generalisation corresponds to a cluster, however it is not necessary that all members of the cluster instantiate that generalisation. Usually it is a subset of these entities which instantiate it. The cluster coverage metric is defined to assess this aspect. This metric, along with the number of instantiations per generalisation, has been also used for assessing the uniformity of the regularities; however, when RIO does not detect such uniformity in the regularities, further filtering and analysis of the generalisations is needed for isolating regularities and seek specific patterns usually referring to a set of entities covering a particular topic in the ontology.

As described in Chapter 6.4, the Atomic Decomposition can be used for further analysis of the generalisations, as by itself it provides a partitioning of the axioms based on their locality notion. There is also an intuition that a top down parsing of the AD graph can help to define the various topics as these are covered by ontology modules. Such a feature can be helpful for further analysis of the generalisations; they could be used to see how generalisations of the same type distribute over the AD graph and find critical paths that reveal more precise patterns and deviations from these patterns.

All these techniques are fairly novel, thus further investigation is worthy for achieving a better pattern induction.

9.4.5 Evaluation of the Cognitive Effect of Generalisations

This is an aspect of future work which is related with the comprehension of an ontology and the induction of patterns. As mentioned above, the integration of RIO with other methods in the scope of ontology comprehension will require additional user studies for assessing the cognitive impact that patterns can have on the inspection of ontologies. As an initial step towards a more “human friendly” presentation of the regularities, we have presented in Chapter 3.6.1 the labeling of clusters, thus corresponding variables in the generalisation with a meaningful name, derived from common characteristics of the clustered entities, whenever this was possible. Such naming could facilitate the inspection of regularities and provide summaries of the axioms and entailments in the ontology with respect to the detected generalisations. However, additional studies and experiments would shed light on how users can be guided from the generalisations for the comprehension of the ontology.

9.4.6 Reasoner Optimisation and Benchmarking

Two additional directions of RIO application are related with reasoner performance. First, an application of RIO could be a pattern-based optimization of reasoners by reducing the terminology size when checking for logical errors, which will make both authoring and using these ontologies feasible. RIO can be used for the generation of ontology templates populating "artificial" ontologies as described in [TGC07, Š07]. The key idea is to use the generalisations for extending the schema of an ontology. This will enable the usage of a less randomised artificial ontology generation, which is useful for evaluating the performance of reasoners.

9.4.7 Implementation Improvements

This section discusses improvements of the implemented methods that can be added as features in future work.

Algorithm Optimisations

Clustering is the computational bottleneck of the RIO framework. In the work presented in this thesis, we applied hierarchical agglomerative clustering for detecting similarities between entities in an ontology. The selected clustering algorithm is described in the literature and is a simple and standard approach for performing unsupervised detection of patterns. The clustering algorithm used in RIO has been implemented and tailored to the nature of the problem, thus no external library was used; the reason for this was to focus more effort on the preprocess and post-process of data than in the knowledge discovery procedure. Although the extensive empirical analysis has shown that RIO can detect meaningful regularities in ontologies, it is worth investigating different clustering algorithms and use existing libraries to enhance both results and performance.

Implemented Tools

The algorithms described in this thesis have all been implemented, and verified during the evaluation of the results, although a number of implementation details are of prototypical quality. For example, a prototype visualization tool already exists, and has been presented to the community [MIS12]; however, to improve utility and diffusion, various features should be included, starting with the addition of support to show semantic regularities: the current implementation can,in fact, only visualise syntactic regularities.

The current implementation of the RIO plugin for Protégé 4, a standalone Java tool, needs to be run beforehand for generating clusters and generalisations for an ontology. The results, in the form of an XML file, can then be loaded and visualised in Protégé 4. In future tasks, the clustering implementation will be integrated with the visualisation plugin for Protégé 4.

9.5 Overall Conclusion

Together RIO and the work on quality assurance lay the foundations of the infrastructure necessary for systematic inspection and quality assurance of ontologies, through the use of ontology reverse engineering methods like the ones presented and evaluated in this thesis, and the future directions outlined in this Chapter.

Bibliography

[AAKS08] Mikel Egaña Aranguren, Erick Antezana, Martin Kuiper, and Robert Stevens. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC bioinformatics*, 9(Suppl 5):S1, 2008.

[AB06] Harith Alani and Christopher Brewster. Metrics for ranking ontologies. In *4th Int'l. EON Workshop, 15th Int'l World Wide Web Conf*, 2006.

[Ara09] Mikel Egana Aranguren. *Role and Application of Ontology Design Patterns in Bio-Ontologies.* PhD thesis, University of Manchester, 2009.

[BCD$^+$10] Ryan R Brinkman, Mélanie Courtot, Dirk Derom, Jennifer M Fostel, Yongqun He, Phillip Lord, James Malone, Helen Parkinson, Bjoern Peters, Philippe Rocca-Serra, et al. Modeling biomedical experimental processes with obi. *Journal of biomedical semantics*, 1(Suppl 1):S7, 2010.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, D Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[BJSSA05] Andrew Burton-Jones, Veda C. Storey, Vijayan Sugumaran, and Punit Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering*, 55(1):84–102, 2005.

[BM06] Isabelle Bichindaritz and Cindy Marling. Case-based reasoning in the health sciences: What's next? *Artificial intelligence in medicine*, 36(2):127–135, 2006.

[BMW93] Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. The concept assignment problem in program understanding. In *Proceedings of the 15th international conference on Software Engineering*, page 498, 1993.

[BN06] Christopher M. Bishop and Nasser M. Nasrabadi. *Pattern recognition and machine learning.* Springer, New York, 2006.

[BR04] Jonathan BL. Bard and Seung Y. Rhee. Ontologies in biology: design, applications and future challenges. *Nature Reviews Genetics*, 5(3):213–222, 2004.

[Bro81] Daniel Brotsky. Program understanding through cliché recognition. *MIT Artificial Intelligence Laboratory*, 1981.

[BS05] Eva Blomqvist and Kurt Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. *ICEIS (3)*, pages 413–416, 2005.

[BS06] Olivier Bodenreider and Robert Stevens. Bio-ontologies: current trends and future directions. *Briefings in Bioinformatics*, 7(3):256–274, 2006.

[BSP09] Johannes Bauer, Ulrike Sattler, and Bijan Parsia. Explaining by Example: Model Exploration for Ontology Comprehension. *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK*, 2009.

[BVHH$^+$04] Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, Lynn Andrea Stein, et al. OWL web ontology language reference. *W3C recommendation*, 10:2006–01, 2004.

[Cla08] Peter Clark. Knowledge patterns. *Knowledge Engineering: Practice and Patterns*, pages 1–3, 2008.

[Con04] Gene Ontology Consortium. The Gene Ontology (GO) database and informatics resource. *Nucleic acids research*, 32(Database issue):D258, 2004.

[CS09] Ronald Cornet and Stefan Schulz. Relationship groups in SNOMED CT. *Medical Informatics in a United and Healthy Europe*, 150:223–227, 2009.

[CSKD04] Werner Ceusters, Barry Smith, Anand Kumar, and Christoffel Dhaen. Ontology-based error detection in SNOMED-CT. *Proceedings of MEDINFO*, 2004:482–6, 2004.

[DCGPMPSF08] Guadalupe Aguado De Cea, Asunción Gómez-Pérez, Elena Montiel-Ponsoda, and Mari Carmen Suárez-Figueroa. Natural language-based approach for helping in the reuse of ontology design patterns. In *Knowledge Engineering: Practice and Patterns: 16th International Conference, EKAW 2008 Acitrezza, Italy, September 29-October 2, 2008, Proceedings*, page 32, 2008.

[DTI07] Paul Doran, Valentina A. M. Tamma, and Luigi Iannone. Ontology module extraction for ontology reuse: an ontology engineering perspective. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 61–70, 2007.

[EF08] Joerg Evermann and Jennifer Fang. Evaluating Ontologies: Towards a Cognitive Measure of Quality. In *EDOC Conference Workshop, 2007. EDOC'07. Eleventh International IEEE*, pages 109–116, 2008.

[ERSA08] Mikel Egaña, Alan Rector, Robert Stevens, and Erick Antezana. Applying Ontology Design Patterns in Bio-ontologies. In Aldo Gangemi and Jérôme Euzenat, editors, *Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes* *in Computer Science*, pages 7–16. Springer Berlin Heidelberg, 2008.

[FB99] Martin Fowler and Kent Beck. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, Boston, MA, USA, 1999.

[FBIP$^+$10] Jesualdo Tomas Fernandez-Breis, Luigi Iannone, Ignazio Palmisano, Alan L. Rector, and Robert Stevens. Enriching the Gene Ontology via the Dissection of Labels Using the Ontology Pre-processor Language. In *Knowledge Engineering and Management by the Masses*, volume 6317 of *Lecture Notes in Computer Science*, pages 59–73. Springer Berlin / Heidelberg, 2010.

[FLC10] Mariano Fernandez-Lopez and Oscar Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web.* Springer Publishing Company, Incorporated, 2010.

[GA07] Dimitrios Galanis and Ion Androutsopoulos. Generating multilingual descriptions from linguistically annotated OWL ontologies: the NaturalOWL system. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 143–146, 2007.

[Gan05] Aldo Gangemi. Ontology design patterns for semantic web content. *The Semantic Web–ISWC 2005*, pages 262–276, 2005.

[GGM$^+$02] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with dolce. *Knowledge engineering and knowledge management: Ontologies and the semantic Web*, pages 223–233, 2002.

[GHKS07a] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Just the right amount: extracting modules from ontologies. In *Proceedings of the 16th international conference on World Wide Web*, pages 717–726, 2007.

[GHKS07b] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Ontology reuse: Better safe than sorry. *Description Logics*, 250, 2007.

[GHM10] Birte Glimm, Ian Horrocks, and Boris Motik. Optimized Description Logic Reasoning via Core Blocking. In Jürgen Giesl and Reiner Hähnle, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2010)*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 457–471. Springer, 2010.

[GP09] Aldo Gangemi and Valentina Presutti. Ontology design patterns. *Handbook on Ontologies*, pages 221–243, 2009.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

[GPS11] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Categorising logical differences between owl ontologies. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1541–1546. ACM, 2011.

[GPS12] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Ecco: A Hybrid Diff Tool for OWL 2 ontologies. In *Proceedings of OWL: Experiences and Directions Workshop (OWLED) Heraklion, Crete, Greece*, 2012.

[GWS07] Andrew Gibson, Katy Wolstencroft, and Robert Stevens. Promotion of ontological comprehension: Exposing terms and metadata with web 2.0. In *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge at 16th International World Wide Web Conference (WWW2007)*, 2007.

[Han07] David J Hand. Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage by Zdravko Markov, Daniel T. Larose. *International Statistical Review*, 75(3):409–409, 2007.

[HB09] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States*, 2009.

[HBPS08] Matthew Horridge, Johannes Bauer, Bijan Parsia, and Ulrike Sattler. Understanding entailments in OWL. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany*, 2008.

[HBV01] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17:107–145, 2001.

[HJ02] Clyde W. Holsapple and Kshiti D. Joshi. A collaborative approach to ontology design. *Communications of the ACM*, 45(2):42–47, 2002.

[HKR$^+$04] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. *University of Manchester*, 2004.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In *Principles of Knowledge Representation and Reasoning*, pages 57–67, 2006.

[Hla05] Jan Hladik. A Generator for Description Logic Formulas. In *Proceedings of the 2005 International Workshop on Description Logics (DL), Edinburgh, Scotland, UK*, 2005.

[HP10] Matthew Horridge and Bijan Parsia. From justifications towards proofs for ontology engineering. In *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning, KR, AAAI Press*, 2010.

[HPS09a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in OWL. *Proceedings of the 22nd International Workshop on Description Logics (DL), Oxford, UK*, 2009.

[HPS09b] Matthew Horridge and Peter F. Patel-Schneider. OWL 2 Web Ontology Language: Manchester Syntax. *W3C Working Group Note*, 2009.

[HPS10a] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Justification oriented proofs in owl. *The 9th International Semantic Web Conference, (ISWC), Shanghai, China*, pages 354–369, 2010.

[HPS10b] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. *In 7th International Semantic Web Conference, (ISWC), Karlsruhe, Germany*, pages 323–338, 2010.

[HPS11] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The state of bio-medical ontologies. In *Bio-Ontologies*, 2011.

[HS01] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the SHOQ (D) description logic. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 199–204. Lawrence Erlbaum Associates LTD, 2001.

[IERS08] Luigi Iannone, Mikel Egana, Alan Rector, and Robert Stevens. Augmenting the expressivity of the ontology pre-processor language. *OWL Experiences and Directions (OWLEd 2008). Karlsruhe, Germany: OWL Experiences and Directions*, 2008.

[IPRS10] Luigi Iannone, Ignazio Palmisano, Alan L. Rector, and Robert Stevens. Assessing the safety of knowledge patterns in owl ontologies. *The Semantic Web: Research and Applications*, pages 137–151, 2010.

[IRS09] Luigi Iannone, Alan L. Rector, and Robert Stevens. Embedding knowledge patterns into OWL. *The Semantic Web: Research and Applications*, pages 218–232, 2009.

[JHI$^+$10] Simon Jupp, Matthew Horridge, Luigi Iannone, Julie Klein, Stuart Owen, Joost Schanstra, Robert Stevens, and Katy Wolstencroft. Populous: A tool for populating ontology templates. *CoRR*, abs/1012.1745, 2010.

[JKS$^+$11] Simon Jupp, Julie Klein, Joost Schanstra, Robert Stevens, et al. Developing a Kidney and Urinary Pathway Knowledge Base. *Journal of biomedical semantics*, 2(Suppl 2):S7, 2011.

[JŁŁ05] Joanna Józefowska, Agnieszka Ławrynowicz, and Tomasz Łukaszewski. Towards discovery of frequent patterns in description logics with rules. *Rules and Rule Markup Languages for the Semantic Web*, pages 84–97, 2005.

[Kee07] C. Maria Keet. Enhancing comprehension of ontologies and conceptual models through abstractions. *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 813–821, 2007.

[Kor02] Bogdan Korel. Computation of dynamic program slices for unstructured programs. *Software Engineering, IEEE Transactions on*, 23(1):17–34, 2002.

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of owl dl entailments. *The Semantic Web, 6th International Semantic Web Conference, ISWC 2007, Busan, Korea*, pages 267–280, 2007.

[Lan05] Chet Langin. *Introduction to Data Mining.* Addison-Wesley, 2005.

[LEL$^+$03] Welf Löwe, Morgan Ericsson, Jonas Lundberg, Thomas Panas, and Niklas Pettersson. Vizzanalyzer-a software comprehension framework. In *3rd Conference on Software Engineering Research and Practise in Sweden*, pages 127–136, 2003.

[LP05] Welf Löwe and Thomas Panas. Rapid construction of software comprehension tools. *International Journal of Software Engineering and Knowledge Engineering*, 15(6):995–1025, 2005.

[MGH$^+$09] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language: Profiles. *W3C Recommendation*, 27:61, 2009.

[MIS12] Eleni Mikroyannidi, Luigi Iannone, and Robert Stevens. RIO: The Regularities Inspector for Ontologies Plugin for Protége-4. In *3rd International Conference on Biomedical Ontology (ICBO)*, June 2012.

[MISR11] Eleni Mikroyannidi, Luigi Iannone, Robert Stevens, and Alan L. Rector. Inspecting regularities in ontology design using clustering. *The Semantic Web–ISWC 2011*, pages 438–453, 2011.

[MISR12] Eleni Mikroyannidi, Luigi Iannone, Robert Stevens, and Alan L. Rector. Inspecting regularities and irregularities in SNOMED-CT. In *Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences*, SWAT4LS '11, pages 76–83, New York, NY, USA, 2012. ACM.

[MJF10] Yinglong Ma, Beihong Jin, and Yulin Feng. Semantic oriented ontology cohesion metrics for ontology-based systems. *Journal of Systems and Software*, 83(1):143–152, 2010.

[MMIS12] Eleni Mikroyannidi, Nor Azlinayati Abdul Manaf, Luigi Iannone, and Robert Stevens. Analysing syntactic regularities in ontologies. In *OWL: Experiences and Directions Workshop, OWLED*, 2012.

[MPSP$^+$09] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. OWL 2 Web Ontology Language: Structural Specification and Functional-style Syntax. *W3C Recommendation*, 27, 2009.

[MRS09] Eleni Mikroyannidi, Alan L. Rector, and Robert Stevens. Abstracting and Generalising the Foundational Model Anatomy (FMA) Ontology. Bio-Ontologies workshop, ISMB conference, 2009.

[MSIR12] Eleni Mikroyannidi, Robert Stevens, Luigi Iannone, and

Alan L. Rector. Analysing Syntactic Regularities and Irregularities in SNOMED-CT. *Journal of Biomedical Semantics*, 3(1):8, 2012.

[MSR11] Eleni Mikroyannidi, Robert Stevens, and Alan Rector. Identifying ontology design styles with metrics. In *7th International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2011.

[MTL78] Robert McGill, John W Tukey, and Wayne A Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.

[NCA08] Natalya Fridman Noy, Abhita Chugh, and Harith Alani. The CKC challenge: Exploring tools for collaborative knowledge construction. *Intelligent Systems, IEEE*, 23(1):64–68, 2008.

[NDG05] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gĭrba. The story of Moose: an agile reengineering environment. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–10, 2005.

[NH97] Natalya Fridman Noy and Carole D. Hafner. The state of the art in ontology design: A survey and comparative review. *AI magazine*, 18(3):53, 1997.

[Pre94] Wolfgang Pree. *Design patterns for object-oriented software development*. Reading, Mass.: Addison-Wesley, 1994.

[PRG$^+$09] Bjoern Peters, Alan Ruttenberg, Jason Greenbaum, Melanie Courtot, Ryan Brinkman, Patricia Whetzel, Daniel Schober, Susanna Assunta Sansone, Richard Scheuerman, and Philippe Rocca-Serra. Overcoming the ontology enrichment bottleneck with quick term templates. *Nature Publishing Group*, 2009.

[PSFGP10] María Poveda, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Common pitfalls in ontology development. In *Current Topics in Artificial Intelligence*, pages 91–100. Springer, 2010.

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In *Proceedings of the 14th international conference on World Wide Web*, pages 633–640, 2005.

[QMFBS12] Manuel Quesada-Martınez, Jesualdo Tomás Fernández-Breis, and Robert Stevens. Enrichment of owl ontologies: a method for defining axioms from labels. *on Capturing and Refining Knowledge in the Medical Domain (K-MED 2012)*, page 1, 2012.

[RBK08] Alan L. Rector, Sebastian Brandt, and Jay Kola. Why do it the hard way? the case for an expressive description logic for snomed. *Journal of the American Medical Informatics Association*, 15(6):744, 2008.

[RBS11] Alan L. Rector, Sam Brandt, and Thomas Schneider. Getting the foot out of the pelvis: modeling problems affecting use of SNOMED CT hierarchies in practical applications. *Journal of the American Medical Informatics Association*, 18(4):432–440, 2011.

[RD00] Ehud Reiter and Robert Dale. *Building natural language generation systems.* Cambridge university press, 2000.

[RDH$^+$04] Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. *Engineering Knowledge in the Age of the SemanticWeb*, pages 63–81, 2004.

[Rec03] Alan L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 121–128, 2003.

[RH07] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 410–420, 2007.

[RI12] Alan L. Rector and Luigi Iannone. Lexically suggest, logically define: Quality assurance of the use of qualifiers and expected results of post-coordination in SNOMED CT. *Journal of Biomedical Informatics*, 45(2):199 – 209, 2012.

[RK02] Juergen Rilling and Bhaskar Karanth. A hybrid program slicing framework. In *Source Code Analysis and Manipulation, 2001. Proceedings. First IEEE International Workshop on*, pages 12–23, 2002.

[RLR98] M. Ramze Rezaee, Boudewijn P. F. Lelieveldt, and Johan H. C. Reiber. A new cluster validity index for the fuzzy c-mean. *Pattern recognition letters*, 19(3):237–246, 1998.

[RMJ$^+$03] Cornelius Rosse, José LV Mejino Jr, et al. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of biomedical informatics*, 36(6):478–500, 2003.

[RRB06] Alan L. Rector, Jeremy Rogers, and Thomas Bittner. Granularity, scale and collectivity: when size does and does not matter. *Journal of Biomedical informatics*, 39(3):333–349, 2006.

[RS02] Juergen Rilling and Ahmed Seffah. MOOSE-a task-driven program comprehension environment. In *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, pages 77–84, 2002.

[RSN08] Daniel L. Rubin, Nigam Shah, and Natalya Fridman Noy. Biomedical ontologies: a functional perspective. *Briefings in bioinformatics*, 9(1):75–90, 2008.

[Sch09] Niels Schütte. Generating natural language descriptions of ontology concepts. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 106–109, 2009.

[SdCH$^+$07] Nicholas Sioutos, Sherri de Coronado, Margaret W. Haber, Frank W. Hartel, Wen-Ling Shaiu, and Lawrence W. Wright. NCI Thesaurus: a semantic model integrating cancer-related clinical and molecular information. *Journal of biomedical informatics*, 40(1):30–43, 2007.

[SDMW02] Kent A Spackman, Robert Dionne, Eric Mays, and Jason Weis. Role grouping as an extension to the description logic of Ontylog, motivated by concept modeling in SNOMED. In *Proceedings of the AMIA Symposium*, page 712, 2002.

[SEA$^+$02] York Sure, Michael Erdmann, Jürgen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. OntoEdit: Collaborative ontology development for the semantic web. *The Semantic Web—ISWC 2002*, pages 221–235, 2002.

[SEM01] Steffen Staab, Michael Erdmann, and Alexander Maedche. Engineering ontologies using semantic patterns. In *Proceedings of the IJCAI-01 Workshop on E-Business & the Intelligent Web*, pages 174–185, 2001.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, pages 26–27, 2008.

[SMW$^+$11] Robert Stevens, James Malone, Sandra Williams, Richard Power, and Allan Third. Automating generation of textual class definitions from owl to english. *Journal of Biomedical Semantics*, 2(Suppl 2):S5, 2011.

[sno11] Snomed technical reference guide. `http://www.ihtsdo.org/fileadmin/user_upload/Docs_01/Publications/SNOMED_CT`, January 2011.

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[SR07] Julian Seidenberg and Alan L. Rector. A methodology for asynchronous multi-user editing of semantic web ontologies. In *Proceedings of the 4th international conference on Knowledge capture*, pages 127–134, 2007.

[ŠZSI10] Ondřej Šváb-Zamazal, Vojtěch Svátek, and Luigi Iannone. Pattern-based ontology transformation service exploiting OPPL and OWL-API. In *Knowledge Engineering and Management by the Masses*, pages 105–119. Springer, 2010.

[TGC07] Yannis Theoharis, George Georgakopoulos, and Vassilis Christophides. On the Synthetic Generation of Semantic Web Schemas. In *Proc. of SWDB-ODBIS*, pages 98–116, 2007.

[TH06a] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: system description. In *Proceedings of the Third international joint conference on Automated Reasoning*, IJCAR'06, pages 292–297, Berlin, Heidelberg, 2006. Springer-Verlag.

[TH06b] Yannis Tzitzikas and Jean-Luc Hainaut. On the visualization of large-sized ontologies. In *Proceedings of the working conference on Advanced visual interfaces*, pages 99–102, 2006.

[TK06] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Third Edition.* Academic Press, Inc., Orlando, FL, USA, 2006.

[TP12] Dmitry Tsarkov and Ignazio Palmisano. Divide et impera: Metareasoning for large ontologies. In *Proc. of 9th International Workshop OWL: Experiences and Directions (OWLED 2012). To Appear*, 2012.

[Tsa12] Dmitry Tsarkov. Improved Algorithms for Module Extraction and Atomic Decomposition. In *Description Logics (DL) Conference*, Rome 2012.

[UHW$^+$98] Mike Uschold, Mike Healy, Keith Williamson, Peter Clark, and Steven Woods. Ontology reuse and application. In *Proceedings of the 1st International Conference on Formal Ontology in Information Systems*, volume 179, page 192, 1998.

[VGK+11] Chiara Del Vescovo, Damian Gessler, Pavel Klinov, Bijan Parsia, Ulrike Sattler, Thomas Schneider 0002, and Andrew Winget. Decomposition and modular structure of bioportal ontologies. In *The Semantic Web – ISWC 2011*, volume 7031 of *Lecture Notes in Computer Science*, pages 130–145. Springer Berlin / Heidelberg, 2011.

[VML02] Anneliese Von Mayrhauser and Stephen Lang. A coding scheme to support systematic analysis of software comprehension. *Software Engineering, IEEE Transactions on*, 25(4):526–540, 2002.

[vMV97] Anneliese von Mayrhauser and A. Marie Vans. Program understanding behavior during debugging of large scale software. In *Papers presented at the seventh workshop on Empirical studies of programmers*, pages 157–179, 1997.

[VPS10] Chiara Del Vescovo, Bijan Parsia, Ulrike Sattler, and Thomas Schneider 0002. The modular structure of an ontology: an empirical study. In *Proceeding of the 2010 conference on Modular Ontologies: Proceedings of the Fourth International Workshop (WoMO 2010)*, pages 11–24, 2010.

[VPS11] Chiara Del Vescovo, Bijan Parsia, Ulrike Sattler, and Thomas Schneider 0002. The modular structure of an ontology: atomic decomposition. In *Proc. of IJCAI*, pages 2232–2237, 2011.

[VPS12] Chiara Del Vescovo, Bijan Parsia, and Ulrike Sattler. Logical relevance in ontologies. In *Description Logics (DL) Conference*, 2012.

[Š07] Šváb, Ondrej and Svátek, Vojtech. In Vitro Study of Mapping Method Interactions in a Name Pattern Landscape. In *Ontology Matching Workshop (OM2007)*, 2007.

[W3C09] W3C OWL Working Group and others. OWL 2 Web Ontology Language: Document Overview. W3C Recommendation. *The World Wide Web Consortium. http://www. w3. org/TR/owl2-overview*, 27 October 2009.

[Wei03] Eric W. Weisstein. *CRC concise encyclopedia of mathematics*. CRC press, 2003.

[WHM$^+$07] Yue Wang, Michael Halper, Hua Min, Yehoshua Perl, Yan Chen, and Kent A. Spackman. Structural methodologies for auditing snomed. *Journal of Biomedical Informatics*, 40(5):561–581, 2007.

[YOE05] Haining Yao, Anthony Mark Orme, and Letha Etzkorn. Cohesion metrics for ontology design and application. *Journal of Computer Science*, 1(1):107–113, 2005.

[YZY06] Zhe Yang, Dalu Zhang, and Chuan Ye. Evaluation Metrics for Ontology Complexity and Evolution Analysis. In *e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on*, pages 162 –170. IEEE Computer Society, 2006.