THE AUSTRALIAN NATIONAL UNIVERSITY

OWL-Miner: Concept Induction in OWL Knowledge Bases

David Ratcliffe

A thesis submitted for the degree of
PhD in Computer Science
The Australian National University

August 2018

Except where otherwise indicated, this thesis is my own original work.

David Ratcliffe
17 August 2018

To my dear wife Natalie,

For all her love, patience, encouragement, and support.

Acknowledgments

I wish to thank:

My supervisor and long-time colleague Kerry Taylor for entrusting me with such an exciting research topic, and for all her support through thick and thin, not only through the course of this thesis, but also during all our work together going back more than 14 years. I simply could not want for a better mentor.

My colleagues and friends Michael Compton, Geoffrey Squire, Michael Kearney and Mark Cameron for numerous discussions, feedback, guidance and encouragement over the many years we had worked together at CSIRO.

Janet Newman, Tom Peat, Vincent Fazio and the rest of brilliant team at the CSIRO Collaborative Crystallisation Centre (C3) and numerous other individuals from the wider protein crystallisation community for guiding me through their fascinating and important science.

My co-supervisors Scott Sanner and Wray Buntine and the Australian National University (ANU) College of Engineering and Computer Science (CECS) AI group including Jochen Renz and John Lloyd for their helpful feedback.

Carole Goble, Alasdair Gray, Frank von Delft, Janna Hastings and Sameer Velankar for hosting my visit to their laboratories in 2011.

The ANU for financial support with an Australian Postgraduate Award and ANU CECS for a top-up scholarship which provided me the freedom and flexibility to pursue this topic. Thanks especially to Janette Rawlinson from CECS for all her support and encouragement.

My friends Adam, Mark, Amit, Rob, Danny, Neil, Jody, Chris and Veena for chats over coffee, single arms, flying mares, and winding hips.

Last but most certainly not least, my wife Natalie for all her love and support, and my parents Graham and Debbie and brother Daniel and the rest of my wonderful family.

Abstract

The Resource Description Framework (RDF) and Web Ontology Language (OWL) have been widely used in recent years, and automated methods for the analysis of data and knowledge directly within these formalisms are of current interest. Concept induction is a technique for discovering descriptions of data, such as inducing OWL class expressions to describe RDF data. These class expressions capture patterns in the data which can be used to characterise interesting clusters or to act as classification rules over unseen data. The semantics of OWL is underpinned by Description Logics (DLs), a family of expressive and decidable fragments of first-order logic.

Recently, methods of concept induction which are well studied in the field of Inductive Logic Programming have been applied to the related formalism of DLs. These methods have been developed for a number of purposes including unsupervised clustering and supervised classification. Refinement-based search is a concept induction technique which structures the search space of DL concept/OWL class expressions and progressively generalises or specialises candidate concepts to cover example data as guided by quality criteria such as accuracy. However, the current state-of-the-art in this area is limited in that such methods: were not primarily designed to scale over large RDF/OWL knowledge bases; do not support class languages as expressive as OWL2-DL; or, are limited to one purpose, such as learning OWL classes for integration into ontologies. Our work addresses these limitations by increasing the efficiency of these learning methods whilst permitting a concept language up to the expressivity of OWL2-DL classes. We describe methods which support both classification (predictive induction) and subgroup discovery (descriptive induction), which, in this context, are fundamentally related.

We have implemented our methods as the system called OWL-Miner and show by evaluation that our methods outperform state-of-the-art systems for DL learning in both the quality of solutions found and the speed in which they are computed. Furthermore, we achieve the best ever ten-fold cross validation accuracy results on the long-standing benchmark problem of carcinogenesis. Finally, we present a case study on ongoing work in the application of OWL-Miner to a real-world problem directed at improving the efficiency of biological macromolecular crystallisation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

1.1.1 Research Goal

The prevalence of Linked Open Data (LOD) under the Resource Description Framework (RDF) data model has grown significantly in recent years as the uptake of Semantic Web technologies continues to increase [13]. The RDF data model and the Semantic Web technologies which are built on top of RDF are relatively new when compared to venerable database technologies such as relational systems, and with them come new challenges around scalability, the design of complex data schemas, and methods for the analyses of data and knowledge in the RDF formalism.

Schema technologies such as the Resource Description Framework Schema (RDFS) and the Web Ontology Language (OWL) are designed for constraining the semantics and structure of RDF data. OWL in particular, being based on highly expressive predicate logics called Description Logics, is capable of expressing complex statements capturing knowledge about data expressed in RDF. However, this expressivity comes at a cost. Not only is the manual construction of OWL ontologies a non-trivial task, but software systems such as automated reasoners for OWL often call for computationally complex algorithms. These two issues pose challenges around the use of OWL as a schema language, and particularly the formal automated analysis of RDF which has been described and constrained with OWL in machine learning and data mining applications. Machine learning and data mining are often already computationally complex in nature, and often require efficient data representations and query mechanisms to aid in the construction of models or detection of patterns.

In recent years, methods have been developed for *automated concept induction*, or the automated construction of OWL classes induced from RDF data [7, 59, 58, 57, 30, 43]. The goals of this body of work appear twofold, firstly to address the complexity of generating OWL classes for ontologies from data and existing background knowledge by automating their construction for addition to ontologies, but secondly to leverage these methods to employ OWL as an expressive language suitable for describing classification rules or cluster descriptions in machine learning and data mining applications. However, current methods around the application of machine learning and data mining with OWL either lack support for highly expressive class languages like OWL2-DL, or are focused on providing support for solving particular kinds of learning problems such as supervised classification. Furthermore, these methods do not support scaling up to solving problems with large amounts of instance data and background knowledge. These limitations hinder the applicability of such learning methods in tackling a broader range of automated learning problems such as subgroup discovery over large knowledge-rich data sets which are becoming more prevalent as semantic technologies are more widely adopted. One such domain where this is particularly the case is in within the life sciences, where large experimental datasets containing rich background knowledge expressed in OWL are being produced, such as the Kidney and Urinary Pathway Knowledge Base (KUPKB) [49].

The goal of our research is to improve the state of the art of automated concept induction in OWL and associated highly expressive DLs in terms of the efficiency and applicability for solving a variety of related supervised machine learning and data mining problems. The techniques we present are designed to leverage rich background knowledge captured as RDFS/OWL over the RDF data being analysed to induce new OWL class expressions. In this way, we treat OWL as a highly expressive hypothesis language, and because the methods we have developed are likely to generate OWL statements which are consistent with background knowledge, they can be used to aid in the construction of OWL ontologies in a semi-automated way. However, our primary focus is to employ OWL as a hypothesis language to support machine learning and data mining tasks in a way which generates comprehensible descriptions of classes and clusters to aid in human interpretation directly as a way of revealing new descriptive knowledge in RDF datasets.

From a practical standpoint, we demonstrate how the methods we have developed to analyse RDF data by mining and classifying with OWL class expressions can be applied to a real-world scientific domain known as biological macromolecular crystallisation. In this domain, several of the world's largest laboratories are working towards capturing scientific experiments using linked RDF data, primarily for the purposes of integration and analysis. Semantic Web technologies such as RDFS and OWL are being used to capture an abundance of relevant background knowledge which can be used to organise and classify this experimental data in interesting ways. In this setting, we show how the application of our concept induction methods over the rich descriptions of scientific experiments can be used to aid scientists in analysing their data with a view to increasing experimental efficiency. This domain is not dissimilar from many other in the life sciences which produce a lot of knowledge-rich data with a need to analyse it with mining and classification techniques. Furthermore, we demonstrate how such domains can use concept induction methods to capture complex axiomatic knowledge as OWL ontologies. This is a useful tool for domain scientists who commonly lack logical modelling skills and may also be a source of new scientific knowledge.

1.1.2 Approach

We begin with a treatise of Description Logics which underpin the semantics of OWL, and describe the state of the art in methods for inducing concepts as DL/OWL expressions from data such as RDF. From there, we identify key limitations with the approaches around scalability, describe how we aim to address these with novel methods while detailing their strengths and drawbacks. We also address the novel problem of subgroup discovery in this setting and implement our methods, demonstrating their efficacy over a well-known problem in this domain. We compare our implementation, the OWL-Miner system, to another comparable existing system known as DL-Learner then we show how OWL-Miner can be used in a real-world setting to address an efficiency problem in the domain of protein crystallisation.

1.1.3 Results

We have implemented our methods in the software system called OWL-Miner and have compared its performance with another state-of-the-art DL learning system, DL-Learner, as covered in Chapter 6. Over a variety of learning problems, we consistently find that our methods not only outperform DL-Learner in terms of efficiency, but at times also in the quality of the solutions found. Efficiency was measured by the number of candidate expressions tested in a search for solutions, as well as system speed. We conclude that our methods do indeed permit a more efficient learning strategy which is suitable for analysing large knowledge bases with highly expressive DLs as the hypothesis language. In Chapter 7, we go on to describe how our system is being integrated into a live system for supporting the analysis of experimental data at CSIRO.

1.2 Thesis Outline

Chapter 2 presents background to the Semantic Web and the problem of knowledge discovery in OWL knowledge-bases, as well as the machine learning and data mining problems we consider in this space. We then describe the particular methods we develop for learning in OWL knowledge-bases around refinement operators, and discuss challenges and the scope of our research in addressing these.

Chapter 3 presents Semantic Web technologies such as the RDF data model, RDF Schema and OWL ontology languages, along with Description Logics which underpin OWL. We then detail formal definitions of aspects of reasoning and learning with DLs, before describing the formal machine learning and data mining settings we consider around supervised classification and subgroup discovery, and present several basic algorithms for learning in relation to these. Finally, we describe the limitations of learning in the formal settings as described, and discuss a more suitable setting for learning around a closed-world interpretation of DLs.

Chapter 4 analyses the state-of-the-art in terms of a refinement operator for OWL, then describes our main contribution which is a novel refinement operator for learning with highly expressive DLs. We also describe a novel extension to the operator to handle numerical data and learning with qualified cardinality restrictions. We summarise the chapter by presenting our novel refinement operator in full, and present an analysis of its properties.

Chapter 5 presents the machine learning and data mining settings we consider and describe novel algorithms for supervised learning which incorporate the novel refinement operator we developed in Chapter 4. We also describe and analyse a class of quality functions for use in learning and describe a novel algorithm for learning in this general setting. We finish by presenting unique modifications to our algorithm to address certain limitations to learning with refinement operators over DLs.

Chapter 6 discusses our implementation, the OWL-Miner system, and presents an evaluation against another state-of-the-art software system, DL-Learner. We compare the performance of these two systems over several benchmark problems before concluding that the OWL-Miner system appears to provide superior performance in terms of search efficiency and of the quality of solutions found.

Chapter 7 presents a case study in the application of the OWL-Miner system to a problem in biology known as protein crystallisation. We begin with background about the domain, and describe current efforts to collect, integrate and describe data using Semantic Web technologies. We then describe how OWL-Miner is being used to aid in the analysis of results to support efficient experimentation, which is ongoing work.

Finally, we summarise our contributions and discuss current and future work when concluding in Chapter 8.

Chapter 2

Background and Related Work

In this chapter we introduce the Semantic Web (§2.1) and the prevalence of new datasets employing Semantic Web technologies to capture data and knowledge, particularly in the domain of life-sciences (§2.1.1). Our motivation is to address the lack of appropriate methods and tools for performing machine learning and data mining over such data sets using Semantic Web technologies directly. We describe the machine learning and data mining problems we will be considering in this thesis (§2.2) and discuss how existing work (§2.3) can be leveraged to develop appropriate techniques in these areas. Finally, we outline the challenges we address in this work to improve the state-of-the-art in machine learning and data mining over OWL knowledge bases (§2.4).

2.1 The Semantic Web

Tim Berners-Lee, widely recognised as the inventor of the World Wide Web, presented a vision of the *Semantic Web* [11, 10] in 2001. The goal was to promote the publishing of machine-interpretable data on the web to enable people and machines to easily find and re-use the data in useful ways. Since then, the World Wide Web Consortium (W3C) has promoted web standards for realising the Semantic Web including technologies such as the Resource Description Framework (RDF) data model, schema (RDFS) and the Web Ontology Language (OWL) [90]. In recent years, the wide uptake of these technologies has seen an abundance of machine interpretable data and knowledge being published on the web, particularly within the life sciences and many other domains including those contributing to the Linked Open Data project [13].

2.1.1 Knowledge Discovery in the Life Sciences on the Semantic Web

As the amount of data and knowledge arriving on the Semantic Web is increasing, there is a clear need to develop machine learning and data mining techniques for direct application to such data. The development of such techniques for the Semantic Web is a current research topic and preliminary studies on the application of existing machine learning and data mining techniques over data captured with the RDF data model and OWL ontologies have been undertaken [97]. Of particular interest is the application of machine learning and data mining methods to knowledge-intensive scientific data on the Semantic Web, particularly in domains where such methods may be used for knowledge discovery to aid in scientific understanding [8].

The use of Semantic Web technologies has had significant uptake in the life sciences, primarily for aiding with data integration and the formal capture of knowledge. Prominent examples include SNOMED-CT [88] and the Gene Ontology (GO) project [103] which aim to provide controlled vocabularies and structured knowledge in the domains of bioinformatics and medicine. Many more ontologies for the life sciences are emerging, such as the ChEBI ontology [25] for capturing chemistry domain knowledge which is core to several fields of bioinformatics including drug discovery [38].

The uptake and abundance of OWL ontologies to describe scientific data presents an unprecedented opportunity to leverage the knowledge captured when performing machine learning and data mining. Specifically, OWL ontologies and their associated terms can be used to describe patterns in the data directly, which make them easily interpretable by domain experts. Furthermore, the formal nature of OWL defines constraints over the structure and relationships of the data it describes, which can be leveraged by machine learning and data mining systems to confine their search spaces for hypotheses and improve their performance, both in terms of efficiency and hypothesis quality.

In our research, we have developed ideas with the design and implementation of novel methods of data mining and machine learning specifically for application directly to OWL knowledge bases. We will describe algorithms for learning and mining which generate hypotheses directly as OWL class expressions over terms from the ontologies used to describe the data being analysed. We will also show how to leverage the structure imposed by OWL to improve the efficiency of the search for hypotheses.

2.2 Machine Learning and Data Mining

Machine learning and data mining are vast and current fields of research. In our work, we focus on the application and analysis of several specific problems and techniques from machine learning and data mining to discovering patterns within OWL knowledge bases. These patterns, when captured as OWL class expressions, aim to provide human readable hypotheses for understanding the reasons behind the production of a pattern. In this way, our approach is similar to *rule learning* which produces hypotheses which can be read by humans and understood in terms of the language of the domain, thus providing insight into the problem being solved. This can be contrast with non-comprehensible learning techniques such as neural networks (NN) or support vector machines (SVM) which are effectively black-box numerical models providing no explanatory capability in terms of the domain of the data.

In this thesis, we are particularly interested in two closely related problems in machine learning and data mining for application to pattern learning in OWL knowledge bases which capture scientific experimental data: classification and subgroup discovery.

2.2.1 Classification

Classification is a supervised machine learning problem which takes a labelled set of examples and seeks to construct hypotheses which differentiate examples based on their labels. In terms of scientific experimental data, examples may be descriptions of experiments where their labels describe the experimental outcome, and hypotheses which differentiate experiments (examples) with one outcome (label) from others can be used to provide insight as to why those experiments have those outcomes. For example, a set of chemical experiments may be labelled with one of *success*, or *failure*. Patterns which exclusively identify those with *failure* and not *success*, e.g., that they contain 50mM sodium acetate, could immediately provide the analyst with the actionable knowledge needed to design successful experiments.

2.2.2 Subgroup Discovery

Subgroup discovery is a supervised data mining problem which takes a labelled set of examples and seeks to construct hypotheses which describe collections of examples with a statistically unusual distribution of labels relative to some baseline [109].

For instance, examples may correspond to chemical experiments evenly partitioned into (success, failure). An interesting hypothesis may describe a large subgroup of all experiments (examples) which predominantly contains successful experiments (75%) with the remainder being failed experiments (25%). As this distribution of examples is significantly different to the set of all experiments (50% success/50% failure), the hypotheses describing the subgroup may indicate *why* the particular set of examples it describes are mostly successful. In contrast with the supervised classification problem above, subgroup discovery is useful when it is unnecessary or impossible to construct individual patterns which capture one label exclusively over another. In this way, subgroup discovery is used as a *descriptive* technique for human consumption, as opposed to the *prescriptive* approach of classification which can be used to construct predictive models.

2.3 Learning OWL Classes

In this thesis, we focus on methods for learning new OWL class expressions which describe example data in some way. This is a general technique which can be applied to describe patterns in sets of example data for the purposes of machine learning and data mining as described earlier (§2.2). For example, a set of learned OWL classes may act as a predictive model over unseen example instances, or identify interesting clusters or subgroups of example instances.

Various methods already exist for learning OWL class expressions from example data based around inducing expressions in Description Logics (DLs), which underpin the formal semantics of OWL. These methods are closely related to those developed for addressing a similar problem in the different yet related logical formalism of Logic Programs (LP) within the field of Inductive Logic Programming (ILP). The following section (§2.3.1) compares learning in DLs to ILP and highlights various DL learning techniques which were largely motivated by their applicability in ILP.

2.3.1 Comparison with Inductive Logic Programming

Despite the fact that learning in DLs is a new area of research, learning in logic-based formalisms in general is not. The field of Inductive Logic Programming (ILP) is a well-researched area of logic-based relational learning which employs Logic Programs (LP) as the formalism for capturing data, background knowledge and hypotheses. There are several key areas in which ILP differs to learning in DLs, including:

- **Standards and uptake.** The W3C recommended RDF, RDFS, OWL and XML Schema are widely used web standards which have enjoyed significant uptake to describe data and knowledge from many domains, particularly in the life sciences. Existing data and ontologies published in these formalisms may be leveraged directly to enrich and structure one's own data for the purposes of learning in a DL. In contrast, background knowledge and data as logic programs as used in ILP are not as widely accessible for these purposes.
- **Expressivity.** ILP algorithms cannot be applied directly to learning in DLs because of the mismatch in the syntax and semantics of the logics [14]. ILP systems typically employ a Horn or definite clause hypothesis and background knowledge representation based on Logic Programs (LP). DLs are different in that they permit complex concepts including positive disjunction, full negation and qualified cardinality restrictions which cannot naturally be expressed in LP, but which may be used in a DL concept based hypothesis language. LP systems can express multiple horn-clause definitions for predicates in learning, which enables them to capture more complex background knowledge than a DL knowledge-base. However, depending on the scale of the data, the results of such complex processing can nevertheless be captured in a DL-knowledge base.
- **Generality.** As semantic entailment of clauses in ILP is undecidable, the syntactic θ-subsumption notion of generality over clauses in ILP is often used as a decidable substitute for comparing the generality or specificity of clausal hypotheses. In many DLs, a natural notion of generality is that of concept subsumption based on model inclusion for which decidable algorithms are known[1]. Concept subsumption in DLs is a semantic notion of generality which, when constrained by terminological background axioms (TBox), naturally constrain the space of permissible hypotheses in meaningful ways. This may be contrasted with syntactic θ-subsumption in ILP where, unless *mode declarations* [68] are used to control the instantiation of variables in a tight way, many irrelevant hypotheses may be permissible which can unnecessarily inflate the search space.
- **Complexity.** Datalog clause coverage and subsumption (determined with θ-subsumption) in ILP are NP-complete problems [37]. Alternatively, certain DLs

[1]In fact, most DLs are specifically designed to ensure that satisfiability is decidable. This is important as the commonly used deductive inference tasks such as concept subsumption and instance checking are reducible to satisfiability checking.

such as $\mathcal{EL}^{++}$ permit PTime reasoning procedures for the analogous tasks of instance checking and concept subsumption [4]. Despite such tasks in expressive DLs having worse computational complexity (e.g. $\mathcal{ALC}$ for which such tasks are PSpace-complete), highly optimised reasoning strategies exist which make them tractable in practice [35]. A direct comparison of the complexity of these tasks is not straightforward, as concept subsumption in a DL knowledge base is usually performed with respect to the entire TBox[2], whereas Datalog clause subsumption in ILP can be performed in a pairwise manner without respect to background clauses.

- **Language bias.** For particular DLs, some language bias is captured directly within the language and may be imposed by a TBox. For instance, the notion of *type* bias in ILP [68] may be is captured with domain and range constraints on roles expressible directly in the language of DLs, restricting their applicability to certain classes. Additionally, mode declarations in ILP are irrelevant when learning in DLs as DLs are variable-free.
- **Closed vs. open world assumption.** Typically, ILP systems will assume a *closed world* (CWA), whereby all facts or data not currently known to the system are assumed to be false. In contrast, DLs in the context of OWL typically make the assumption of an *open world* (OWA), whereby no logical conclusions may be drawn from facts or data which are not currently known. With OWL, which is underpinned by DLs, an open world suits the nature of the intended application which is to describe data on the web, not all of which is feasible to capture in any one system for analysis such as logical reasoning. However, the chosen assumption has direct implications on the method and computational complexity of logical reasoning and related tasks such as retrieving the set of data described by a logical expression. In ILP, such tasks are usually tractable in a closed world as they involve algorithms of low computational complexity. In DLs which assume an open world, especially those of high expressivity, such reasoning tasks can have extremely high computational complexity which poses practical limitations on the size of any knowledge base and data set.

Despite these differences, several fundamental techniques developed in ILP research are relevant to learning in DLs. We now describe two of the most influential techniques in sections 2.3.2 and 2.3.3 which have motivated our work.

[2]However, note that *incremental classification* is a technique which can be used to test for concept subsumption which does not require the re-classification of the entire TBox [20].

2.3.2 Non-standard inferences

Deductive inference problems in DLs are well-studied, including concept subsumption and instance checking. More recent research into so-called *non-standard inferences* in DLs include techniques for generating concept expressions from instance data via the *most specific concept* (msc) procedure (similar to constructing the so-called bottom clause with saturation in ILP) and the *least common subsumer* (lcs) for concept expressions without disjunction (analogous to the *least general generalisation* (lgg) of clauses in ILP) [5]. The LCSLearn algorithm for the DL C-Classic [18] employs the msc and lcs for inducing concepts from instance data in the way the ILP system GOLEM [67] learns clauses in a bottom-up manner. SONIC is a recent implementation of algorithms for computing the msc and lcs in the more recent $\mathcal{EL}^{++}$ language [106].

2.3.3 Refinement operators

Motivated by techniques in ILP, *refinement operators* for generalising or specialising hypotheses to traverse the hypothesis search space have been researched for a number of DLs including $\mathcal{ALER}$ [7], $\mathcal{EL}$ [57] and even highly expressive DLs such as $\mathcal{SROIQ}(D)$ which underpins OWL2-DL [58]. Implementations of DL learners which implement a top-down search with refinement operators include DL-Learner [56] and DL-FOIL [30], the latter of which implements a covering (separate-and-conquer) approach for the DL $\mathcal{ALC}$ based on the well-known FOIL algorithm in ILP [78]. YinYang [43, 29] is another DL learner which combines top-down and bottom-up refinement search also in $\mathcal{ALC}$. Each of these implementations are designed to induce a single concept expression which classifies instance data with high accuracy. Fr-ONT [55] uses top-down refinement for the DL $\mathcal{EL}$ for discovering frequent patterns in a DL knowledge base akin to the Warmr [48] algorithm for data mining in ILP. Recently, DL-Learner was extended [73] to learn in the probabilistic DL known as cr$\mathcal{ALC}$ [19].

Systems which perform concept induction exclusively in the formalism of DLs using refinement based search have been described recently. Most notably, DL-Learner [56] is a system for DL concept learning over highly expressive DLs supporting supervised classification and unsupervised learning. Indeed many of our methods and our implementation, the OWL-Miner system, were designed around improvements to the methods employed by DL-Learner. OWL-Miner differs from DL-Learner in its ability to support a variety of data mining and machine learning tasks including subgroup discovery. Also, while the search procedure that DL-Learner employs is based on refinement, it is not guided by the distribution of data in the knowledge base in the way we have developed the specialisation operator for OWL-Miner. Other work on DL concept induction for supervised classification include the YinYang [43] and DL-FOIL [30] systems which employ the less expressive DLs $\mathcal{ALC}$ in their hypothesis languages. Fr-ONT is another implementation of a concept learner in the DL known as $\mathcal{EL}^{++}$, roughly corresponding to the OWL-EL profile, and is designed to compute frequent queries in a data mining setting [55]. In contrast, OWL-Miner uses a more expressive hypothesis language, permits highly expressive background knowledge, and is capable of other learning tasks such as subgroup discovery.

2.4 Challenges and Scope

Many challenges remain in developing novel or adapted data mining and machine learning methods for direct application to knowledge discovery over data and knowledge in DL knowledge bases. The key areas which we focus on in this thesis are:

- **Learning under a closed world assumption.** So far, most research has focussed on DL learning settings which rely upon checking the coverage of induced concepts with computationally expensive methods, namely, knowledge-base satisfiability checking via entailment. This makes machine learning and data mining in this setting a difficult prospect, because coverage which is likely to be computed many times in a learning procedure in this way can be prohibitively expensive for highly expressive DLs and large knowledge bases which contain a lot of data and knowledge assertions. Additionally, coverage checking via entailment under an open world assumption may not permit the computation of examples described by expressions which describe *all* data in a domain, such as $\forall r.X$, which states that all examples in the tuples of role r are of type X, which are assumed to be unknown in an open world. These limitations can be overcome by learning in a closed world setting, which is traditionally employed in data mining and machine learning. However, learning which assumes a closed world over a knowledge base and constraints defined with open world semantics poses challenges. Of these challenges, the most significant is that of correct coverage computation, and the induction of satisfiable concept expressions with respect to the background knowledge base.

- **Efficiently learning qualified cardinality restrictions (QCRs).** Qualified cardinality restrictions (such as the concept expression $\geqslant^4 r.X$ describing how any instance in the domain of r must have at least four r-successors) pose significant challenges in DL learning as they vastly expand the search space. Optimised methods for handling learning in the presence of QCRs in the hypothesis language, such as when learning expressive OWL classes, are needed.
- **Learning in DLs with various concrete domains.** Research into learning in DLs has so far largely focussed on concept induction exclusively over abstract domains without handling concrete domains such as numerical data. Scientific data on the Semantic Web includes numerical data, and learning procedures which cope with numerical features together with abstract data are required for knowledge discovery in this context. Preliminary work on handling concrete domains with refinement operators has been developed and implemented in DL-Learner [56, 104], however many data mining methods based on search with refinement operators have yet to be explored over DL knowledge bases with concrete domains.
- **Subgroup discovery.** Techniques for performing subgroup discovery over OWL knowledge bases are interesting as they can have the potential to be aided by high-quality background knowledge of various forms [3] for improving hypothesis quality and restricting the hypothesis search space. While related work exists on identifying frequent patterns in $\mathcal{EL}$ knowledge-bases with the Warmr algorithm [55], very little work exists in the context of subgroup discovery in DL knowledge bases which can be useful sources of highly expressive background knowledge. In many scientific domains, such analyses are also useful for gaining insights into experimental data sets.
- **Efficient algorithms for mining large DL knowledge bases.** Research into DL learning has so far focussed primarily on theoretical aspects such as learnability, various component techniques such as refinement operators and proof-of-concept implementations of the application of various techniques from ILP as outlined in section 2.3. However, efficient algorithms for mining knowledge bases using DLs with a large amount of data and background knowledge have not yet been explored. This focus is motivated by the observation that knowledge bases employing RDF and OWL are now capable of handling millions to billions of RDF statements. Results in this area are generally applicable to mining of data on the Semantic Web, a current research topic [97], and are ap-

plicable in real-world domains in which vast amounts of data produced as RDF and described with OWL can be analysed with automated mining and machine learning techniques. The domain of biological macromolecular crystallisation, which we explore in a case study in Chapter 7, is an exemplar.

While many challenges remain, we are motivated to focus on the aforementioned areas which are specifically applicable to knowledge discovery in the life sciences on the Semantic Web. In particular, we address an important real-world problem in structural biology, *protein crystallisation*, for which experimental data and knowledge are being made available with Semantic Web technologies which has motivated our work. Protein crystallisation is typical of other life sciences with data and knowledge on the Semantic Web in that it is a highly knowledge-intensive domain which seeks to use RDF and OWL primarily for the purposes of data integration and mining [71]. Furthermore, protein crystallisation is a field which can benefit greatly from *comprehensible* knowledge discovery tools to aid scientists in gaining a deeper understanding of the field. Throughout the thesis, we will use examples from the domain of protein crystallisation, and provide a case-study into the application of the techniques developed in this work in Chapter 7.

2.5 Summary

In this chapter, we have introduced the Semantic Web (§2.1) and the uptake of Semantic Web technologies in capturing important datasets and knowledge-bases in the life sciences (§2.1.1). The current lack of tools for performing machine learning and data mining over these datasets is the motivation for the work in this thesis. We discussed how our work is focussed on developing three useful and general machine learning and data mining techniques (§2.2) for application to analysing OWL knowledge bases, and existing work in this area (§2.3). We then described the state-of-the-art in learning in OWL knowledge bases and highlighted the primary areas which we address to progress work in this area (§2.4).

Chapter 3

Preliminaries

In this chapter we present the basic formalisms around the data models, logics and learning methods which are used throughout this thesis. While most of these preliminaries presented in this chapter reflect established work as referenced throughout, Section 3.5 describes a novel closed-world setting for learning with DLs which we build upon for our other novel contributions in DL learning in subsequent chapters.

3.1 The RDF Data Model

3.1.1 The Resource Description Framework: RDF

The *Resource Description Framework* (RDF) is a widely-used model for capturing graph-based data which is recommended by the World Wide Web Consortium (W3C) for describing data on the web. RDF prescribes the use of International Resource Identifiers (IRIs) to identify resources, and relates these with other IRIs representing certain relationships between them. For example, consider the following:

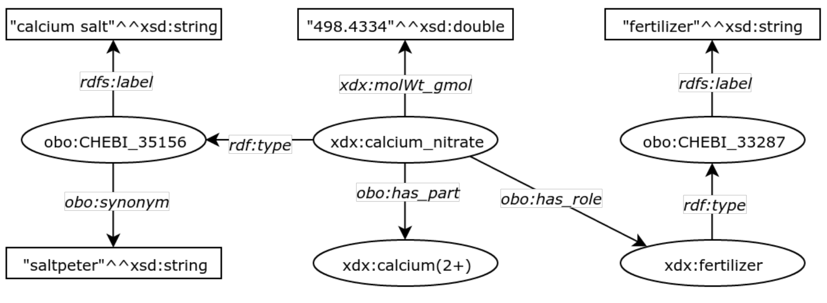


Figure 3.1: An example RDF graph describing information about the chemical calcium nitrate. The graph combines resources from various namespaces including the ChEBI ontology for describing chemical compounds. It describes calcium nitrate as belonging to the class of calcium salts labelled *CHEBI_35156*, also known as *saltpeter*, as having a molecular weight of 498.4334 g/mol and a part which is a calcium(2+) ion, and having a role of fertilizer.

In Figure 3.1, resources are identified with ellipses, directed arcs represent named properties (also by IRI) which relate resources. Rectangles represent *literals* which are XSD Schema types, such as strings or doubles. An RDF database corresponding to a graph consists of a collection of *triples*, which are tuples of the form *(subject, predicate, object)*. Table 3.1 represents several triples from the graph of Figure 3.1.

```
(xdx:calcium_nitrate,  rdf:type,        obo:CHEBI_35156)
(xdx:calcium_nitrate,  xdx:molWt_gmol,  "498.4334"^^xsd:double)
(xdx:calcium_nitrate,  obo:has_role,    xdx:fertilizer)
(obo:CHEBI_35156,      rdfs:label,      "calcium salt"^^xsd:string)
...                    ...              ...
```

Table 3.1: A partial set of RDF triples corresponding to parts of the graph from Figure 3.1.

In this way, RDF is a flexible data model which can be used to describe arbitrary objects, relate objects together via properties, and attribute objects to concrete data such as numbers or strings. RDF is associated with two other W3C recommended standards which describe *schema languages* for RDF graphs, which are discussed in the next section.

3.1.2 RDF Schema Languages: RDFS, OWL

Elements of an RDF graph may be described and constrained in various ways using two particular schema languages known as RDF Schema (RDFS) and the Web Ontology Language (OWL).

3.1.2.1 RDFS

RDFS is a schema language which can be used to categorise and constrain RDF resources in several basic ways. Firstly, RDFS can define *classes* which describe the type of a collection of RDF resources. For example, `obo:CHEBI_35156` is an RDF class (which is is itself an RDF resource) which represents *"the class of all things which are calcium salts"*. In Figure 3.1, the resource `xdx:calcium_nitrate` is asserted to be a member of this class via the `rdf:type` property.

RDFS can also be used to capture relationships between RDFS classes, such as the fact that `obo:CHEBI_35156` is a *subclass* of the more general class `xdx:Compound`, or `obo:CHEBI_33287` (the class of fertilizers) is a *subclass* of `xdx:Role`, as shown in Figure 3.2. Such relationships can be used to construct detailed class hierarchies

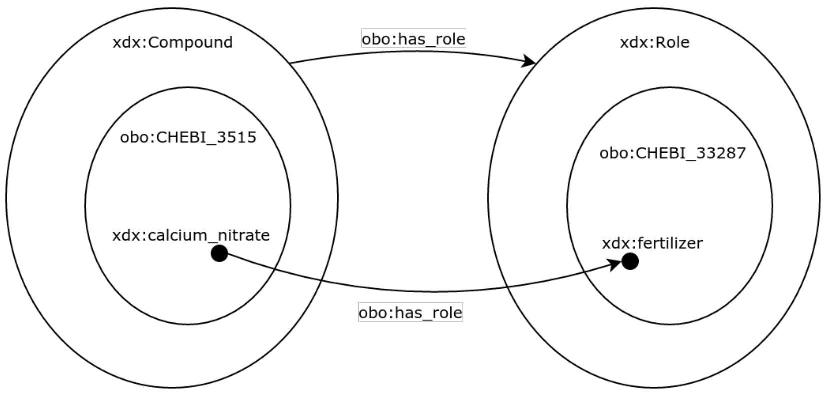


Figure 3.2: An abstract RDFS diagram representing the class hierarchy between `obo:CHEBI_35156` (calcium salts) and all chemical compounds `xdx:Compound`, and where the resource `xdx:calcium_nitrate` is an instance of both. The property relationship `obo:has_role` is shown between `xdx:Compound` and `xdx:Role` to describe the domain and range of this property, and its usage is shown between the class instances `xdx:calcium_nitrate` and `xdx:fertilizer`.

which capture knowledge about a domain like chemistry. In this case, the resource `xdx:calcium_nitrate` could be *inferred* to also belong to the class `xdx:Compound`. Methods of inference used to determine this are described in detail in subsequent sections of this chapter.

The use of RDF properties can also be constrained with RDFS by defining the applicable classes of resources to be used in some property's domain and range. For example, the property `obo:has_role` may be constrained to relate resources which are of class `xdx:Compound` to those of class `xdx:Role`.

RDFS can also describe how some property may be more specific than another, for example, by denoting that `xdx:hasIon` is a *subproperty* of `obo:has_part`. With such an assertion, all triples with the property `xdx:hasIon` may be inferred to also belong to the property `obo:has_part`.

3.1.2.2 OWL: The Web Ontology Language

OWL brings much more expressivity than RDFS for describing restrictions on classes and properties over RDF data and RDFS classes and properties. As it is the focus of this thesis to employ OWL as the primary hypothesis language for learning, it is introduced and treated in more detail in the next section.

3.2 OWL and Description Logics

The Web Ontology Language (OWL) is a schema language for RDF that extends RDFS with a variety of expressive language constructs for defining classes, properties and their relationships to capture knowledge about domains. In addition to RDFS constructs such as subclass, sub-property, and domain and range assertions, OWL can be used to define complex restrictions on class definitions.

3.2.1 Description Logics

OWL is underpinned by *Description Logics* (DL) [6], a family of knowledge representation languages based on decidable fragments of first-order logic (FOL) which have well-understood computational properties. The semantics of OWL is expressed directly in terms of the semantics of DLs and has a correspondence to the semantics of RDF[1].

Description Logics have a variable-free syntax for describing *concepts* (e.g. $Com-pound$, $Role$, known as *classes* in RDFS and OWL, corresponding to unary predicates in FOL), *roles* (e.g. $hasPart$, $hasRole$, known as *properties* in OWL, corresponding to binary predicates in FOL), and *individuals* (e.g. $calcium\_nitrate$, corresponding to constants in FOL, or resources in RDF). DLs also describe relationships between these elements, such as the *axiom* $CalciumSalt \sqsubseteq Compound$ to declare that $CalciumSalt$ is a subclass of $Compound$, and the *assertion* $CalciumSalt(calcium\_nitrate)$ to assert that $calcium\_nitrate$ is an instance of the class $CalciumSalt$. Collections of such axioms and assertions together form a DL *knowledge base*, which is often separated into two components: the *TBox* which maintains *terminological* knowledge (concept and role axioms), and the *ABox* which maintains *assertional* knowledge (concept and role instance assertions).

Definition 3.2.1. ***(Knowledge base $\mathcal{K}$)*** *A DL* ***knowledge base*** *is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is the TBox, $\mathcal{A}$ is the ABox.*

Definition 3.2.2. ***(Signature of a Knowledge Base)*** *The* ***signature*** *of a DL knowledge base $\mathcal{K}$ is the triple (N_C, N_R, N_I), where N_C is the set of all concept names, N_R is the set of all role names, and N_I is the set of all named individuals. N_C, N_R and N_I may be considered pairwise disjoint[2].*

[1] http://www.w3.org/TR/owl2-rdf-based-semantics/#Correspondence_Theorem

[2] The technique of *punning* in knowledge bases permits these sets to overlap under certain circumstances, and is supported in OWL2. See: https://www.w3.org/TR/owl2-new-features/#F12:_Punning

Description Logics permit the construction of new concepts with logical constructs which can be used to combine existing concept and role terms. For example, for concept names C, D and role r, conjunction is denoted by $C \sqcap D$, disjunction by $C \sqcup D$, negation with $\neg C$, and existential role restriction as $\exists r.C$ denoting the set of all individuals which have r-successors which are instances of C. Different families of DLs are defined by their inclusion (or exclusion) of various constructs such as these. Two simple DLs which often form the basis of extensions to other DL languages are $\mathcal{ALC}$ (attributive language with complement) and $\mathcal{EL}$ (existential language).

Definition 3.2.3. ***($\mathcal{EL}$ and $\mathcal{ALC}$ concepts)*** *Given a set of concept names N_C and role names N_R, the syntax of $\mathcal{EL}$ and $\mathcal{ALC}$ concepts are defined inductively as follows:*

- *All $A \in N_C$ (atomic concepts), $\top$ (top), and $\bot$ (bottom) are $\mathcal{ALC}$ and $\mathcal{EL}$ concepts;*
- *Where C, D are $\mathcal{EL}$ concepts, the following are also $\mathcal{EL}$ concepts:*
  - *$C \sqcap D$ (conjunction), $\exists r.C$ (existential role restriction)*
- *Where C, D are $\mathcal{ALC}$ concepts, the following are also $\mathcal{ALC}$ concepts:*
  - *$C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), $\neg C$ (negation)*
  - *$\exists r.C$ (existential role restriction), $\forall r.C$ (universal role restriction).*

Various extensions to $\mathcal{ALC}$ and $\mathcal{EL}$ concepts are identified with labels and include the following:

$\mathcal{S}$	$\mathcal{ALC}$ with transitive roles, e.g. $r(i,j), r(j,k) \to r(i,k)$
$\mathcal{H}$	Role hierarchies, e.g. $s \sqsubseteq r$ where $r, s \in N_R$
$\mathcal{R}$	Role composition ($r \circ s \sqsubseteq t$), e.g. $r(i,j), s(j,k) \to t(i,k)$
$\mathcal{O}$	Nominals, e.g. concept $\{i\}$ where $i \in N_I$
$\mathcal{I}$	Inverse roles, e.g. $inv(r(j,i)) \leftrightarrow r(i,j)$
$\mathcal{Q}$	Qualified cardinality restrictions, e.g. ${}^{\geqslant 3}r.C$, ${}^{\leqslant 6}s.D$
$\mathcal{D}$	Concrete domains and datatype roles, e.g. $\exists r.double[\geq 5.6]$

The latest OWL specification (W3C OWL2) describes several *profiles* corresponding to DLs with various levels of expressivity, including EL, RL, QL, and DL. In this work, we primarily consider the highly expressive OWL2-DL profile which corresponds to $\mathcal{SROIQ(D)}$ [42][3].

Definition 3.2.4. ***(DL expressivity ϕ, all concepts $\mathcal{L}_\phi$)*** *The expressivity of a particular DL ϕ is the language consisting of the set of permissible constructs for defining concepts,*

[3]See http://www.w3.org/TR/owl2-direct-semantics for more details.

assertions and axioms. The expressivity implies a set of all possible concepts expressible within that DL, denoted $\mathcal{L}_{\phi}$.

OWL permits so-called *concrete domains* to describe elements other than abstract individuals, such as numbers, strings and boolean values, as well as other user-defined types. In the OWL specification, concrete domains include those which are modelled to permit restrictions over literals of the various XML Schema types permissible in RDF graphs. These restrictions are known as *facets* which are operators ranging over a set of concrete values, such as the expression $int[> 5]$, representing all integers $[6, \infty)$, and boolean combinations thereof, such as $double[(\geqslant 5 \wedge <6) \vee (> 6 \wedge <7)]$ representing all double values in the ranges $[5, 6)$ and $(6, 7)$. This language gives us the ability to capture subsets of numbers in the concrete domains of integers or doubles.

The *semantics* of DL concepts are defined by a first-order *interpretation* over a set of elements called the *domain of interpretation* Δ that maps concept expressions to subsets of Δ, and roles to pairs of elements of Δ.

Definition 3.2.5. ***(Interpretation)*** *An* ***interpretation*** *is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ defined over the signature (N_C, N_R, N_I) of a knowledge base (Definition 3.2.2) where $\Delta^{\mathcal{I}}$ is a non-empty set, and $\cdot^{\mathcal{I}}$ maps:*

- *Concepts $C \in N_C$ to a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$;*
- *Roles $r \in N_R$ to a subset $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$;*
- *Individuals $i \in N_I$ to an element $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.*

The syntax and semantics of various DL concept constructs are shown in Table 3.2. Similarly, the semantics of concrete domains $\mathcal{D}$ are defined as follows.

Definition 3.2.6. ***(Concrete domain $\mathcal{D}$)*** *A* ***concrete domain*** *$\mathcal{D}$ describes a domain set $\Delta^{\mathcal{D}}$ and a set of predicates $pred(\mathcal{D})$, known as the predicate names of $\mathcal{D}$. Each predicate name $P \in pred(\mathcal{D})$ is associated with an arity n and an n-ary predicate $P^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$.*

Example 3.2.7. *The concrete domains $\mathcal{R}$, $\mathcal{Z}$ and $\mathcal{N}$ each respectively represent the set of all reals $\Delta^{\mathcal{R}} = \mathbb{R}$, integers $\Delta^{\mathcal{Z}} = \mathbb{Z}$, and non-negative integers $\Delta^{\mathcal{N}} = \mathbb{N}$. The set pred over these domains each contains the binary predicate names $<, \leq, \geq, >$.*

Construct	Syntax	Semantics
Concept assertion	$C(i)$	$i^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertion	$r(i,j)$	$\langle i^{\mathcal{I}}, j^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$
Top	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Bottom	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
Negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Nominal	$\{i\}$	$\{i\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, \sharp\{i\}^{\mathcal{I}} = 1$
Conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential role restriction	$\exists r.C$	$\{x \mid \exists y.\langle x,y \rangle \in r^{\mathcal{I}} \land y \in C^{\mathcal{I}}\}$
Universal role restriction	$\forall r.C$	$\{x \mid \forall y.\langle x,y \rangle \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
Min. quantified role restriction	${}^{\geqslant n}r.C$	$\{x \mid \sharp\{y.\langle x,y \rangle \in r^{\mathcal{I}} \land y \in C^{\mathcal{I}}\} \geqslant n\}$
Max. quantified role restriction	${}^{\leqslant n}r.C$	$\{x \mid \sharp\{y.\langle x,y \rangle \in r^{\mathcal{I}} \land y \in C^{\mathcal{I}}\} \leqslant n\}$
Exact quantified role restriction	${}^{=n}r.C$	$\{x \mid \sharp\{y.\langle x,y \rangle \in r^{\mathcal{I}} \land y \in C^{\mathcal{I}}\} = n\}$

Table 3.2: The syntax and semantics of $\mathcal{ALCOQ}$ concepts where C, D are concepts, r is a role name, i, j are individuals, and $\sharp S$ denotes the cardinality of set S.

Example 3.2.8. *Consider the following interpretation $\mathcal{I}_1$ over concepts representing chemical compounds and elements:*

$$\begin{array}{rcll}
\Delta^{\mathcal{I}_1} & = & \{x, na_0, cl_0, & \\
 & & y, mg_0, cl_1, cl_2, & \\
 & & z, mg_1, cit_0\} & \\
Na^{\mathcal{I}_1} & = & \{na_0\} & \\
Cl^{\mathcal{I}_1} & = & \{cl_0, cl_1, cl_2\} & \\
Mg^{\mathcal{I}_1} & = & \{mg_0, mg_1\} & \\
Citrate^{\mathcal{I}_1} & = & \{cit_0\} & \\
hasPart^{\mathcal{I}_1} & = & \{\langle x, na_0\rangle, \langle x, cl_0\rangle, & \textit{(x: sodium chloride)} \\
 & & \langle y, mg_0\rangle, \langle y, cl_1\rangle, \langle y, cl_2\rangle, & \textit{(y: magnesium dichloride)} \\
 & & \langle z, mg_1\rangle, \langle z, cit_0\rangle\} & \textit{(z: magnesium citrate)}
\end{array}$$

Given this interpretation, we can define the following concepts:

$$(\exists hasPart.\top)^{\mathcal{I}_1} = \{x, y, z\} \textit{ (chemical compounds)}$$

$$(\exists hasPart.Cl)^{\mathcal{I}_1} = \{x, y\} \textit{ (chloride salts)}$$

Consider the concept:

$$({}^{\geqslant 2}hasPart.Cl)^{\mathcal{I}_1} = \{y\} \textit{ (salts with } \geq 2 \textit{ chloride ions)}$$

Note that under $\mathcal{I}_1$, we find that $cl_1^{\mathcal{I}_1} \neq cl_2^{\mathcal{I}_1}$, as otherwise there are not at least two chlorine atoms for individual y to be an instance of $(\geqslant^2 hasPart.Cl)$. Consider a different interpretation, $\mathcal{I}_2$, which is defined similarly to $\mathcal{I}_1$ except that it maps $cl_1^{\mathcal{I}_2} = cl_2^{\mathcal{I}_2}$, resulting in $(\geqslant^2 hasPart.Cl)^{\mathcal{I}_2} = \emptyset$. While atypical for DLs generally, the DLs which underpin OWL do not make assumptions about the distinctness of differently named individuals, however the unique name assumption may be used to ensure this.

Definition 3.2.9. ***(Unique Name Assumption)*** *An interpretation $\mathcal{I}$ respects the* ***unique name assumption (UNA)*** *if, for any two named individuals with different names $i \neq j$, they are mapped to different elements where $i^{\mathcal{I}} \neq j^{\mathcal{I}}$.*

As described in Table 3.2, an interpretation $\mathcal{I}$ can also be applied to the TBox which consists of a set of axioms of the form $C \sqsubseteq D$ called *inclusion axioms*.

Definition 3.2.10. ***(Interpretation satisfies/models TBox)*** *An interpretation $\mathcal{I}$* ***satisfies*** *(is a* ***model*** *of) a TBox $\mathcal{T}$ (written as $\mathcal{I} \models \mathcal{T}$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each axiom $C \sqsubseteq D$ in $\mathcal{T}$.*

Definition 3.2.11. ***(Consistency)*** *A knowledge base $\mathcal{K}$ is said to be* ***consistent*** *iff there exists at least one interpretation which is a model of $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$).*

Definition 3.2.12. ***(Subsumes)*** *A concept D* ***subsumes*** *concept C (written as $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$. A concept D* ***strictly subsumes*** *concept C (written as $C \sqsubset D$) iff $C^{\mathcal{I}} \subset D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$. (Strict) subsumption in the context of a TBox $\mathcal{T}$ is denoted $C \sqsubseteq_{\mathcal{T}} D$ ($C \sqsubset_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ ($C^{\mathcal{I}} \subset D^{\mathcal{I}}$) holds for all models $\mathcal{I} \models \mathcal{T}$.*

Definition 3.2.13. ***(Equivalent)*** *A concept C is* ***equivalent*** *to concept D (written as $C \equiv D$) iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$. Equivalence in the context of a TBox $\mathcal{T}$ is denoted $C \equiv_{\mathcal{T}} D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ holds for all models $\mathcal{I} \models \mathcal{T}$. Note that $C \equiv_{\mathcal{T}} D$ is equivalent to the case where both $C \sqsubseteq D$ and $D \sqsubseteq C$ are in $\mathcal{T}$.*

Definition 3.2.14. ***(Instance)*** *An individual $i \in \Delta^{\mathcal{I}}$ is called an* ***instance*** *of concept C if $i^{\mathcal{I}} \in C^{\mathcal{I}}$.*

Definition 3.2.15. ***(Interpretation satisfies/models ABox)*** *An interpretation $\mathcal{I}$* ***satisfies*** *(is a* ***model*** *of) an ABox $\mathcal{A}$ (written as $\mathcal{I} \models \mathcal{A}$) if, for all individual assertions $\phi \in \mathcal{A}$, we have $\mathcal{I} \models \phi$ where:*

- *$\mathcal{I} \models C(i)$ if $i^{\mathcal{I}} \in C^{\mathcal{I}}$;*
- *$\mathcal{I} \models r(i,j)$ if $\langle i^{\mathcal{I}}, j^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$.*

Definition 3.2.16. ***(Concept satisfiability)*** *A concept C is said to be* ***satisfiable*** *iff there exists at least one interpretation $\mathcal{I}$ where $C^{\mathcal{I}} \neq \emptyset$. C is said to be* ***satisfiable wrt*** *$\mathcal{T}$ iff there exists at least one model $\mathcal{I} \models \mathcal{T}$ where $C^{\mathcal{I}} \neq \emptyset$.*

Definition 3.2.17. ***(Interpretation satisfies/models knowledge base)*** *An interpretation $\mathcal{I}$* ***satisfies*** *(is a* ***model*** *of) a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ (written as $\mathcal{I} \models \mathcal{K}$) iff $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$.*

Definition 3.2.18. ***(Entailment)*** *A knowledge base $\mathcal{K}$ is said to entail some statement ϕ (written as $\mathcal{K} \models \phi$) iff, for all interpretations $\mathcal{I}$ which are models of $\mathcal{K}$ (where $\mathcal{I} \models \mathcal{K}$), we have $\mathcal{I} \models \phi$. For example, if $\mathcal{K} \models C(i)$, then it must hold that $i^{\mathcal{I}} \in C^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ which are models of $\mathcal{K}$ (where $\mathcal{I} \models \mathcal{K}$).*

There may be multiple interpretations which satisfy (are models of) $\mathcal{T}$, $\mathcal{A}$, or $\mathcal{K}$. Furthermore, there are subtleties in the interpretation of asserted knowledge which we illustrate in Example 3.2.19.

Example 3.2.19. *Consider the follow knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where:*

$$\begin{aligned}
\mathcal{T} &= \{Inorganic \equiv \neg Organic, \\
&\quad Na \sqcup Cl \sqcup Mg \sqsubseteq Inorganic, \\
&\quad Citrate \sqsubseteq Organic, \\
&\quad Compound \sqsubseteq \exists hasPart.\top\} \\
\mathcal{A} &= \{Na(na_0), Cl(cl_0), Cl(cl_1), Cl(cl_2), \\
&\quad Mg(mg_0), Mg(mg_1), Citrate(cit_0), \\
&\quad hasPart(x, na_0), hasPart(x, cl_0), \\
&\quad hasPart(y, mg_0), hasPart(y, cl_1), hasPart(y, cl_2), \\
&\quad hasPart(z, mg_1), hasPart(z, cit_0)\}
\end{aligned}$$

The interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ of Example 3.2.8 are models of both $\mathcal{A}$ and $\mathcal{T}$, as they matche all assertions in $\mathcal{A}$, and satisfy all axioms in $\mathcal{T}$, therefore $\mathcal{K}$ is consistent. Note that $\mathcal{K} \not\models (^{\geqslant 2}hasPart.Cl)(y)$, as not all interpretations which are models of $\mathcal{K}$ entail this (namely, $\mathcal{I}_2$ does not). Furthermore, consider the concept:

$$(\forall hasPart.Inorganic)$$

Because $(Na \sqcup Cl \sqcup Mg \sqsubseteq Inorganic) \in \mathcal{T}$, we might expect that

$$\mathcal{K} \models (\forall hasPart.Inorganic)(x) \text{ and } \mathcal{K} \models (\forall hasPart.Inorganic)(y)$$

as these do not have inorganic parts asserted to $\mathcal{A}$. However, both of these entailments would not follow in DL knowledge bases which make the open world assumption. This is because it is assumed that there may exist other hasPart-successors of x or y which are not inorganic, but are simply currently unknown, such as $\langle x, cit\rangle \in hasPart^{\mathcal{I}}$.

Example 3.2.19 highlights the important fact that typically, DL knowledge bases make the *open world assumption*. This choice is made to reflect the desire that knowledge bases should have *incomplete* knowledge of the world, and to minimise the chance that the addition of new assertions do not render the knowledge base inconsistent. However, in Example 3.2.19, we may want to assume that $\mathcal{A}$ contains *all* information relevant to a domain, and that further assertions are not possible. In this case, we may wish to make the *closed world assumption* (CWA), which permits us to assume that our knowledge base is complete. If we use the CWA, we effectively restrict the set of models $\mathcal{I} \models \mathcal{K}$ to those where no more assertions exist beyond $\mathcal{A}$ and any entailments from $\mathcal{T}$, such that in Example 3.2.19 we would find that $\mathcal{K} \models (\forall hasPart.Inorganic)(x)$ and $\mathcal{K} \models (\forall hasPart.Inorganic)(y)$. This is an important concept for machine learning and data mining which we will explore in more detail in Section 3.5.

3.2.1.1 Reasoning Tasks in Description Logics

Given a knowledge base, the set of assertions in $\mathcal{A}$ comprise the base set of explicit information asserted about a domain, such as that certain named individuals represent chemical compounds. When combined with a non-empty set of axioms in $\mathcal{T}$, more assertions may be *implied*, such as in Example 3.2.8 which showed how certain instances belong in the interpretation of complex concepts like those defining sodium salts. *Inference algorithms* for DL reasoning are employed to compute such implicit information within DL knowledge bases, and are often implemented for particular DL languages. Implementations of such algorithms are called DL *reasoners*. For example, the CEL system[4] is designed to compute inferences for the DL $\mathcal{EL}+$, and Pellet[5] is designed to compute inferences for several DLs including $\mathcal{EL}$ up to $\mathcal{SROIQ}$. There are several standard inference tasks for DLs which include *terminological reasoning* such as: checking for knowledge base consistency; concept subsumption and equivalence; and *assertional reasoning* which focuses on individuals, such as *instance checking* and *retrieval*.

[4]CEL reasoner homepage: http://lat.inf.tu-dresden.de/systems/cel/
[5]Pellet reasoner homepage: http://clarkparsia.com/pellet

Definition 3.2.20. ***(Instance Checking)*** *For some knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$,* ***instance checking*** *is the problem of verifying if, for some individual $i \in N_I$ and concept C, that i is an instance of C for all interpretations $\mathcal{I}$ of $\mathcal{K}$, where $i^{\mathcal{I}} \in C^{\mathcal{I}}$ (denoted $\mathcal{K} \models C(i)$). Note that if $\mathcal{K} \models \neg C(i)$, we may conclude that i is not an instance of concept C (denoted $\mathcal{K} \not\models C(i)$), as otherwise $\mathcal{K}$ would be inconsistent.*

Definition 3.2.21. ***(Instance Retrieval)*** *For some knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$,* ***instance retrieval*** *is the problem of determining the set of instances of some concept C, denoted $R_{\mathcal{K}}(C) = \{i \in N_I \mid \mathcal{K} \models C(i)\}$.*

Definition 3.2.22. ***(Classification of a Knowledge Base)*** *For some knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$,* ***classification*** *is the problem of determining all entailments of the form $\mathcal{K} \models (C \sqsubseteq D)$, where C, D are concepts in the signature N_C of $\mathcal{K}$.*

The task of classification is often used to re-organise the subsumption hierarchy of all concepts in the signature N_C of a knowledge base by determining where each concept sits in terms of subsumption to every other concept in N_C. Generally, instance checking and retrieval are often tied to the problem of classification which must be performed to ensure completeness of the entailments. Classification and retrieval can be computationally expensive procedures in practice, as they both require analysis of the relationships potentially between each concept in N_C and individual in N_I respectively.

Implementations of reasoning algorithms over DL knowledge bases are often designed to achieve certain desirable computational properties, including *soundness*, *completeness* and *decidability*.

Definition 3.2.23. ***(Soundness)*** *Given a knowledge base $\mathcal{K}$, an inference algorithm which derives conclusion ϕ from $\mathcal{K}$ (written as $\mathcal{K} \vdash \phi$) is said to be* ***sound*** *iff, for all inferred conclusions, $\mathcal{K} \models \phi$ (all conclusions are* ***valid****).*

Definition 3.2.24. ***(Completeness)*** *An inference algorithm is said to be* ***complete*** *iff for a knowledge base $\mathcal{K}$ and some conclusion ϕ where $\mathcal{K} \models \phi$, it is also true that the algorithm will derive $\mathcal{K} \vdash \phi$.*

Definition 3.2.25. ***(Decidability)*** *Given a knowledge base $\mathcal{K}$ and any statement ϕ, $\mathcal{K}$ is said to be* ***decidable*** *if there exists an algorithm $\vdash$ which can compute whether $\mathcal{K} \models \phi$ holds which always terminates.*

These properties are highly desirable, as soundness ensures that inferences are valid; completeness ensures that all possible entailments with respect to the semantics of the DL will be inferred; and decidability ensures that inference will never get stuck in a loop and will always terminate. Unfortunately, meeting all of these properties together often comes at a cost, as most inference algorithms for expressive DLs which are sound, complete and decidable are computationally expensive. For example, the DL $\mathcal{SROIQ}$ is a highly expressive language with most reasoning tasks having N2ExpTime complexity [47]. This means that reasoning over $\mathcal{SROIQ}$ knowledge bases with very many axioms and assertions can quickly become intractable[6]. Often, the source of such complexity is linked to the expressiveness of the DL. The complexity of reasoning in $\mathcal{SROIQ}$ may be contrast with that of the relatively inexpressive DL $\mathcal{EL}^{++}$ which underpins OWL2-EL for which most inference tasks have PTime complexity [4].

Most inference tasks including classification, instance checking and retrieval are reducible to the problem of satisfiability checking (model checking) in a DL knowledge base depending on the particular DL language used [6]. As such, instance retrieval for any concept C in knowledge base $\mathcal{K}$ may require pre-classification of $\mathcal{K}$, which can be highly computationally expensive.

3.3 Machine Learning and Data Mining

In this work, we aim to describe methods for machine learning and data mining over data and knowledge maintained in DL knowledge bases. In particular, we are concerned with methods for generating new DL concept expressions as *hypotheses* which describe patterns in the data. In this section, we describe the particular settings in machine learning and data mining which we will address in this thesis, and in Section 3.4 we describe how we apply techniques for learning concepts in DL knowledge bases to these settings.

3.3.1 Supervised Learning Problems

Supervised learning problems typically take a set of examples $\mathcal{E}$ for which each member $e_\omega \in \mathcal{E}$ has been attributed with some label $\omega \in \Omega$ where $|\Omega| \geq 2$. In this way, the set of examples can be partitioned into sets containing examples with a common label ω, denoted $\mathcal{E}^\omega$ where $\mathcal{E} = \bigcup_{\forall \omega \in \Omega} \mathcal{E}^\omega$. In a supervised learning problem, we seek to construct *hypotheses* h which describe certain proportions of each of the labelled examples of each set $\mathcal{E}^\omega$. We say that a hypothesis *covers* an example, denoted

[6]However, highly optimised reasoner algorithms do exist to handle moderately large knowledge bases, such as the hyper-tableaux calculus implemented by the HermiT system [91]

by the boolean function $covers(h, e_\omega)$, if h describes example e_ω where $e_\omega \in \mathcal{E}^\omega$. With respect to the set of all examples $\mathcal{E}$, we denote the *cover* of hypothesis h as the set $cover(h, \mathcal{E}) = \{e \in \mathcal{E} \mid covers(h, e)\}$.

3.3.1.1 Classification

The typical *binary classification* problem in machine learning has two labels $|\Omega| = \{+, -\}$, where $\mathcal{E}^+$ are the *positive* examples and $\mathcal{E}^-$ are the *negative* examples. Hypotheses are sought which cover all *positive* examples $\forall e \in \mathcal{E}^+ : covers(h, e)$ and none of the *negative* examples $\forall e \in \mathcal{E}^- : \neg covers(h, e)$.

The performance of any hypothesis h in a learning problem is often assessed with a *measure function* f which maps hypotheses h from the set of all possible hypotheses $\mathcal{L}$ and their covers $cover(h, \mathcal{E}) \subseteq \mathcal{E}$ to real values.

Definition 3.3.1. ***(Measure Function)*** *Given a set of labelled examples $\mathcal{E}$ and the space of all hypotheses $\mathcal{L}$, a* ***measure function*** *is a real-valued function $f : \mathcal{L} \times \{\mathcal{E}\} \mapsto \mathbb{R}$ which maps pairs of hypotheses $h \in \mathcal{L}$ and the set of labelled examples $\mathcal{E}$ to a real value denoting the performance of h over $\mathcal{E}$.*

In order to describe when a hypothesis h may be considered a *solution* to a learning problem based on its cover over a set of examples $\mathcal{E}$, we define a threshold τ over f where $f(h, \mathcal{E}) \geq \tau$ describes h as being a solution. A *quality function* is a boolean function which succeeds when h is a solution in terms of some threshold τ on a measure function f.

Definition 3.3.2. ***(Quality Function)*** *Given a set of labelled examples $\mathcal{E}$ and the space of all hypotheses $\mathcal{L}$, a* ***quality function*** *is a boolean function $\mathcal{Q} : \mathcal{L} \times \{\mathcal{E}\} \mapsto \mathbb{B}$ which maps pairs of hypotheses $h \in \mathcal{L}$ and the set of labelled examples $\mathcal{E}$ to a boolean value denoting whether h may be considered a solution to a learning problem over $\mathcal{E}$. Quality functions are often defined in terms of a minimum threshold τ over a measure function f where $\mathcal{Q}(h, \mathcal{E}) = f(h, \mathcal{E}) \geq \tau$.*

An example of a commonly used measure function for assessing hypothesis performance in binary classification is *accuracy*, which is defined as follows.

Definition 3.3.3. ***(Accuracy)*** *Given a labelled set of examples $\mathcal{E}$ where $\mathcal{E} = \bigcup_{\omega \in \Omega} \mathcal{E}^\omega$ and $\Omega = \{+, -\}$ partitioned into positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, a hypothesis h and its cover C where $C = cover(h, \mathcal{E})$, the* ***accuracy*** *function is defined as:*

$$acc(h, \mathcal{E}) = \frac{TP + TN}{TP + FP + FN + TN}$$

where

$$\begin{array}{rcll} TP & = & |\mathcal{E}^+ \cap C| & \textit{(true positives)} \\ FP & = & |\mathcal{E}^- \cap C| & \textit{(false positives)} \\ TN & = & |\mathcal{E}^- \setminus C| & \textit{(true negatives)} \\ FN & = & |\mathcal{E}^+ \setminus C| & \textit{(false negatives)} \end{array}$$

A quality function over accuracy may be defined as $\mathcal{Q}(h,\mathcal{E}) : acc(h,\mathcal{E}) > 0.95$ *which holds when hypothesis* h *has an accuracy over* 95%*.*

As hypotheses in classification problems are often sought to exclusively cover examples of a single common label, they can be used for *prediction*. Given a new unseen, unlabelled example u, we may use a hypothesis h which is deemed a solution to label u by testing if $covers(h,u)$ succeeds. The performance of a hypothesis considered a solution for a classification problem relative to a set of labelled examples $\mathcal{E}$, also known as the *training set*, can be tested with a *test set* of unseen labelled examples, $\mathcal{U}$. Any hypothesis h which was induced over a training set of labelled examples $\mathcal{E}$ can then be assessed for performance over unseen test data by computing its measure f relative to $\mathcal{U}$. If h performs well over $\mathcal{E}$ and $\mathcal{U}$, we may consider h to be a good classification hypothesis suitable for *prediction*, and may use it to provide labels for new examples. If h performs poorly over $\mathcal{U}$, then it may be considered a poor predictor. In this case, h may have been induced to *over-fit* the set of training data $\mathcal{E}$ such that is does not *generalise* well to previously unseen examples $\mathcal{U}$. One approach to assessing whether h will generalise well to unseen examples is to split the set of examples $\mathcal{E}$ composed of labelled sets $\mathcal{E}^{\omega}$ for each $\omega \in \Omega$ into $k \geq 2$ training and test set pairs $(\mathcal{E}_i,\mathcal{U}_i)$ for $1 \leq i \leq k$, where $\mathcal{E}_i \subset \mathcal{E}$ and $\mathcal{U}_i = \mathcal{E} \setminus \mathcal{E}_i$, and where each pair $(\mathcal{E}_i,\mathcal{U}_i)$ is composed of roughly the same proportion of labelled examples relative to $\mathcal{E}$. By training h on each $\mathcal{E}_i$, we assess its performance on the remaining examples $\mathcal{U}_i$, and compute the overall performance as the average measure over each set $f(h,\mathcal{U}_i)$. This technique is known as *cross validation*. Partitioning the training set into k different sets is called *k-fold cross validation*, as we generate k different 'folds' of test and training data. Other techniques also exist for ensuring that hypotheses generalise well, such as by ensuring the expressions they are composed of are as simple as possible, according to the minimum description length principle [84] which formalises *Occam's razor* in that "among competing hypotheses, the one with the fewest assumptions should be selected".

3.3.1.2 Subgroup Discovery

Subgroup discovery is another interesting supervised learning problem [40]. Subgroup discovery differs from classification in that hypotheses are intended only to be *descriptive* of the examples they cover, and are not generally expected to be used for prediction. Typically, a hypothesis which is considered a solution to a subgroup discovery problem covers an interesting or unusual distribution of labelled examples relative to the distribution in a population set of examples $\mathcal{E}$. Typically, so-called *correlation measures* are used to determine the performance of hypotheses which measure the amount of deviation in the distribution of labelled examples relative to the population. An example of a commonly used correlation function in subgroup discovery is the *weighed relative accuracy* measure, which is defined below.

Definition 3.3.4. ***(Weighed Relative Accuracy)*** *The weighed relative accuracy correlation measure is a real-valued function $\sigma_{wra} : \mathcal{L} \times \mathcal{E} \to \mathbb{R}$ which maps pairs $(C, \mathcal{E})$ for some concept $C \in \mathcal{L}$ and set of binary labelled examples $\mathcal{E}$ where $\mathcal{E} = \bigcup_{\forall \omega \in \Omega} \mathcal{E}^{\omega}$ for $\Omega = \{+, -\}$ to a real value as follows:*

$$\sigma_{wra}(h, \mathcal{E}) = \frac{|cover(h, \mathcal{E}^+)|}{|\mathcal{E}^+|} - \frac{|cover(h, \mathcal{E}^-)|}{|\mathcal{E}^-|}$$

Example 3.3.5. *Consider two sets of labelled examples, $\mathcal{E}^+$ and $\mathcal{E}^-$, where $|\mathcal{E}^+| = 50$ and $|\mathcal{E}^-| = 50$. Also consider two hypotheses h_0, h_1 where:*

- *h_0 covers 42 examples, where 3 are from $\mathcal{E}^+$ and 49 are from $\mathcal{E}^-$;*
- *h_1 covers 46 examples, where 45 are from $\mathcal{E}^+$ and 1 are from $\mathcal{E}^-$.*

We define our quality function $\mathcal{Q}(h, \mathcal{E})$ which succeeds if h is sufficiently interesting to be considered a solution as $\mathcal{Q}(h, \mathcal{E}) = |\sigma_{wra}(h, \mathcal{E})| > 0.9$, which represents a significant deviation from the set of example labels amongst $\mathcal{E}^+$ and $\mathcal{E}^-$.

$$\begin{aligned}\sigma_{wra}(h_0, \mathcal{E}) &= |\tfrac{3}{50} - \tfrac{49}{50}| = 0.92 \\ \sigma_{wra}(h_1, \mathcal{E}) &= |\tfrac{45}{50} - \tfrac{1}{50}| = 0.88\end{aligned}$$

Therefore, we conclude that hypothesis h_0 is interesting, while h_1 is not.

3.4 Learning in DL Knowledge Bases

Thus far, we have described DLs (§3.2) and several settings for learning (§3.3) which we aim to apply to learning in DL knowledge bases by generating DL concept expressions as hypotheses. The method of learning DL concepts we will focus on is known as *induction*. Induction seeks to construct new hypotheses which explain or describe example data. In a DL knowledge base, induction means learning DL concept expressions which describe (or exclude) certain individuals, subject to constraints such as background knowledge and quality criteria. In this way, DL concepts induced as hypotheses in a learning problem are intended to reveal new structural knowledge about the individuals they cover.

Definition 3.4.1. ***(Concept Induction)*** *Concept induction in a knowledge base $\mathcal{K}$ is the problem of computing new (complex, non-atomic) concept expressions C where, for all concepts D in the signature of $\mathcal{K}$, we have $C \not\equiv D$.*

In a learning problem which seeks to induce DL concepts as hypotheses, we require a method which generates candidate concept expressions for testing, along with methods for testing their coverage over examples. We begin by describing methods of searching for candidates (§3.4.1), describe operators used for searching a space of concepts called *refinement operators* (§3.4.2), and describe how they can be used (§3.4.3). We then conclude the section with a discussion on how hypothesis cover is computed in DL knowledge bases, along with some important limitations (3.4.4).

3.4.1 Learning as Search for Concepts

In searching for DL concepts as hypotheses in a learning problem, we are required to generate candidate expressions from the space of all concepts within some DL language $\mathcal{L}$. A basic method for achieving this is known as the *generate-and-test* method as shown in Algorithm 1. This method enumerates every possible DL concept h from the space of concepts $\mathcal{L}$, and tests if each h is a solution to a learning problem over the set of all examples $\mathcal{E}$ with a boolean quality function $\mathcal{Q}(h, \mathcal{E})$ which succeeds only if h can be considered a solution relative to the set of examples $\mathcal{E}$. We denote the space of DL concepts which can be composed of concept and role names from knowledge base $\mathcal{K}$ with expressivity $\mathcal{L}$ as $\mathcal{L}_{\mathcal{K}}$.

For most DLs, the number of possible concepts in $\mathcal{L}_{\mathcal{K}}$ may be large or unbounded, so the enumeration of all concepts for testing by Algorithm 1 is practically infeasible. Instead, we seek to structure the space of concepts in a way which may permit a search to be orderly and efficient. Concept subsumption ($\sqsubseteq$) is one such way of structuring the space of DL concepts appropriate for this purpose. This technique is used in the field of Inductive Logic Programming (ILP) for structuring expressions

Algorithm 1 A basic *generate-and-test* method which enumerates all concept expressions in $\mathcal{L}_{\mathcal{K}}$ and tests them for sufficient quality over all examples $\mathcal{E}$ with the boolean quality function $\mathcal{Q}(h, \mathcal{E})$.

```
1: $S = \emptyset$                                  ▷ The set of solutions
2: for all $C \in \mathcal{L}_{\mathcal{K}}$ do
3:     if $\mathcal{Q}(C, \mathcal{E}) = true$ then
4:         $S := S \cup \{C\}$
5:     end if
6: end for
```

in Logic Programs. In ILP, methods for traversing a structured space of expressions have been explored with functions called *refinement operators*. Recently, research into refinement operators in ILP have been carried over to DLs, and various refinement operators for a number of DLs which underpin OWL have been studied [59, 58, 57]. We now describe refinement operators (§3.4.2) which reproduces Definitions 3.4.2 to 3.4.8 from this existing body of work, then we will describe how to integrate refinement operators into search-based algorithms for learning (§3.4.3).

3.4.2 Refinement Operators

The set of all DL concept expressions for some language $\mathcal{L}$ can be considered as being ordered by the subsumption relationship, $\sqsubseteq$. As the relation $\sqsubseteq$ is a *quasi-order* in that it is reflexive ($C \sqsubseteq C$) and transitive ($A \sqsubseteq B$ and $B \sqsubseteq C$ imply $A \sqsubseteq C$), we can define a *quasi-ordered space* of DL concepts as the pair $(\mathcal{L}, \sqsubseteq)$. A *refinement operator* is a function which is designed to traverse concepts in this ordered space.

Definition 3.4.2. ***(Refinement Operator)*** *Given a quasi-ordered space $(\mathcal{L}, \sqsubseteq)$, a* ***refinement operator*** *is a mapping from $\mathcal{L}$ to $2^{\mathcal{L}}$ where, $\forall C \in \mathcal{L}$, we have:*

- $\rho(C) : \{D \mid D \in \mathcal{L} \wedge D \sqsubseteq C\}$ *(a* ***downward*** *refinement operator);*
- $\upsilon(C) : \{D \mid D \in \mathcal{L} \wedge D \sqsupseteq C\}$ *(an* ***upward*** *refinement operator)*

where each $D \in \rho(C)$ are called ***specialisations*** *of C, and each $D \in \upsilon(C)$ are called* ***generalisations*** *of C.*

Definition 3.4.3. ***(Closure of a Refinement Operator)*** *Given a quasi-ordered space $(\mathcal{L}, \sqsubseteq)$, a refinement operator τ and some $C \in \mathcal{L}$, we define the* ***closure*** *of τ for C, denoted $\tau^*(C)$, as:*

$$\tau^*(C) = \tau^0(C) \cup \tau^1(C) \cup \ldots \cup \tau^n(C) \cup \ldots$$

where $\tau^0(C) = \{C\}$ and $\tau^n(C) = \{D \mid \forall E \in \tau^{n-1} : D \in \tau(E)\}$.

Refinement operators permit the search for concepts by traversing the space in a stepwise manner with repeated application on refined concepts. For example, from C we can reach $D \in \tau(C)$, and from D we can reach $E \in \tau(D)$, and so on, in a *chain* of refinement steps.

Definition 3.4.4. ***(Refinement Chain, Reachability, Passes Through)*** *A* ***refinement chain*** *of length $n \geq 1$ with refinement operator τ is a finite sequence of refinements of individual concepts from C_0 as $C_1 \in \tau(C_0)$, then $C_2 \in \tau(C_1)$, ..., then $C_n \in \tau(C_{n-1})$, otherwise denoted by $C_0 \rightsquigarrow_\tau C_1 \rightsquigarrow_\tau C_2 \rightsquigarrow_\tau \ldots \rightsquigarrow_\tau C_{n-1} \rightsquigarrow_\tau C_n$. If it is possible to construct a refinement chain from concept C to some other concept D, we say that D is* ***reachable*** *from C. If a refinement chain contains some concept C, we say it* ***passes through*** *concept C.*

Furthermore, we can describe a number of useful properties which characterise the behaviour of a particular refinement operator function τ. For example, we may wish that a refinement operator produces true specialisations or generalisations of some concept, and not concepts which are equivalent to the input which may otherwise cause loops in the search, wasting computational resources.

Definition 3.4.5. ***(Proper, Improper Refinement)*** *A refinement operator τ is* ***proper*** *if, for any $D \in \tau(C)$, it holds that $D \not\equiv C$. Properness ensures that τ generates true specialisations $D \sqsubset C$ (generalisations $D \sqsupset C$) of some concept C. Otherwise, we refer to the refinement as being* ***improper****.*

Another desirable property of refinement operators is that they only produce a finite number of concepts in the refinement of any concept.

Definition 3.4.6. ***(Locally Finite Refinement)*** *A refinement operator τ is* ***locally finite*** *if, for any $C \in \mathcal{L}$, $\tau(C)$ is finite and computable.*

When using downward (upward) refinement operator τ to reach concepts within $\mathcal{L}$, we may want to ensure that if some concept $D \sqsubseteq C$ ($D \sqsupseteq C$), that it is reachable in the closure of $\tau^*(C)$. This property is known as *completeness* and permits a search for concepts with ρ to be sure that if a concept $D \sqsubseteq C$ ($D \sqsupseteq C$), it can be reached with τ from C.

Definition 3.4.7. ***(Complete Refinement)*** *A downward (upward) refinement operator ρ (δ) is* ***complete*** *if $\forall C, D \in \mathcal{L}$, $D \sqsubseteq C$ ($D \sqsupseteq C$) implies that $E \in \rho^*(C)$ ($E \in \delta^*(C)$) where $E \equiv D$.*

Definition 3.4.8. ***(Redundant Refinement)*** *A refinement operator τ is* ***redundant*** *if, from any concept C, it admits at least two refinement chains: $C \rightsquigarrow_\tau \ldots \rightsquigarrow_\tau D \rightsquigarrow_\tau \ldots \rightsquigarrow_\tau E$ which does not go through concept F, and $C \rightsquigarrow_\tau \ldots \rightsquigarrow_\tau F \rightsquigarrow_\tau \ldots \rightsquigarrow_\tau E'$ where $E' \equiv E$. We say that τ is* ***non redundant*** *if it only ever admits a single unique refinement chain between any two non-equivalent concepts.*

Redundancy is undesirable because a search for concepts with a redundant operator τ may encounter the same concept more than once, potentially wasting computational resources. In the next section, we will describe basic algorithms which use refinement operators to search a space of DL concepts for learning.

3.4.3 Concept Induction by Refinement-Based Search

While Algorithm 1 demonstrated the basics of a generate-and-test search approach, it did not define how to generate concepts for testing. We now describe how this can be achieved through the use of refinement operators as described in Section 3.4.2. Algorithm 2 demonstrates this using a downward refinement operator, ρ.

Algorithm 2 A basic *generate-and-test* method which uses a refinement operator ρ to search the space of concepts $(\mathcal{L}, \sqsubseteq)$, and a quality function $\mathcal{Q}$ to assess hypothesis performance over the dataset $\mathcal{E}$. A frontier list L is maintained for all candidates to be searched.

```
1:  $L = [\top]$                              ▷ The hypothesis frontier of candidates
2:  $S = \emptyset$                            ▷ The set of solutions
3:  while $length(L) > 0$ do
4:      $C := pop(L)$                          ▷ Remove $C$ from $L$
5:      if $\mathcal{Q}(C, \mathcal{E}) = true$ then
6:          $S := S \cup \{C\}$                ▷ Hypothesis $C$ is a solution
7:      else
8:          for all $C' \in \rho(C)$ do
9:              $push(C', L)$                  ▷ Add refinement $C'$ to $L$
10:         end for
11:     end if
12: end while
```

Algorithm 2 demonstrates how a refinement operator can be incorporated into the search to learn hypotheses. This is a general purpose algorithm which searches a space of concepts in $\mathcal{L}$ from the top concept $\top$ and progressively specialises expressions added to a list. If, on line 9, a candidate concept is added to the head the of list, the search proceeds *depth-first*, whereas if it is added to the tail of the list, the search proceeds *breadth-first*.

The structuring of the search space permits certain quality functions to have the property that if a hypothesis h does not pass a quality test, then neither will all of its (upward, downward) refinements. This property is known as *(anti-)monotonicity*.

Definition 3.4.9. ***(Monotonicity, Anti-monotonicity)*** *For all hypotheses $C, D \in \mathcal{L}$ and the set of all examples $\mathcal{E}$, a quality function $\mathcal{Q}$ is known as* ***monotonic*** *iff*

$$\forall E \subseteq \mathcal{E} : (C \sqsubseteq D) \wedge \mathcal{Q}(D, E) \rightarrow \mathcal{Q}(C, E)$$

and $\mathcal{Q}$ is known as ***anti-monotonic*** *iff*

$$\forall E \subseteq \mathcal{E} : (C \sqsubseteq D) \wedge \mathcal{Q}(C, E) \rightarrow \mathcal{Q}(D, E)$$

[23].

(Anti-)monotonic functions are useful as they permit us to prune potentially large parts of the search space away. Once a hypothesis h fails by an (anti-) monotonic quality function, then we can safely exclude, or *prune*, all (specialisations) generalisations of h from the search by not considering (downward) upward refinements of h. An example of an (anti-)monotonic quality function is *relative frequency*.

Definition 3.4.10. ***(Relative Frequency)*** *Given a hypothesis C and a set of examples $\mathcal{E}$,* ***relative frequency*** *is defined as $relFreq(C, \mathcal{E}) = \frac{|cover(C, \mathcal{E})|}{|\mathcal{E}|}$ [23].*

Example 3.4.11. *Consider the quality function $relFreq(C, \mathcal{E}) \geq t$ where $0 \leq t \leq 1$ as an anti-monotonic quality criterion for downward refinement. Consider an example where $t = 50$, and two hypotheses $C, D \in \mathcal{L}$ where $|cover(C, \mathcal{E})| = 49$. Therefore, C fails the quality function as it does not cover enough examples. By Definition 3.4.9, we know that all concepts $D \in \rho(C)$ refined down from C will never cover more examples than C, so all refinements of C may be excluded from the search.*

By adding (anti-)monotonic quality criteria to a refinement-based search algorithm, we stand to improve the efficiency of the search by excluding hypotheses which can never be considered solutions. Unfortunately, the space of concepts may still be vast even with such pruning, so any frontier list of hypotheses candidates such as that maintained in Algorithm 2 may still grow infeasibly large. One well-known method for dealing with this problem is to simply fix the maximum size of the frontier list, known as *beam search*. A beam search *approximates* the search over all concepts reachable by some refinement operator τ by restricting the search to within a set of candidates. Typically, the restricted size frontier (known as the beam) is only populated with new hypotheses deemed the best relative to the set of refinements of all hypotheses currently maintained in the beam. When the beam is of infinite width, beam search is equivalent to breadth-first search, or best-first search if the hypotheses are ranked within an infinitely sized beam. In order to rank hypotheses a *utility* function is often used, which is often also called a *heuristic evaluation function*.

Definition 3.4.12. ***(Utility Function)*** *A* ***utility function*** $\boldsymbol{u} : \mathcal{L} \times S \mapsto \mathbb{R}$ *maps a pair* $(C, \mathcal{E})$ *where C is a concept expression* $C \in \mathcal{L}$ *for some language* $\mathcal{L}$ *together with a set of examples* $\mathcal{E} \in S$ *and maps it to a real number in* $\mathbb{R}$*. A utility function represents the value of a concept in a learning problem relative to the examples it describes from* $\mathcal{E}$ *and can be used to rank concepts* C, D *such as* $\boldsymbol{u}(C, \mathcal{E}) < \boldsymbol{u}(D, \mathcal{E})$ *which indicates that concept D is preferred over C. Utility functions are often based on measures, for example accuracy (Definition 3.3.3) or relative frequency (Definition 3.4.10).*

By ordering elements of a beam relative to a utility function **u**, we can maintain the list of current best n candidates of the search. In this way, **u** acts as a *heuristic* by permitting the search to proceed into parts of the space of concepts deemed most likely to contain solutions to the exclusion of other parts. Depending on the size of the beam and behaviour of the heuristic, a search may reach solutions faster, yet it may also exclude subsets of concepts from the space which contain the best solutions. This is why such methods are known to be *approximate*, as they are not complete, and may inadvertently confine a search into a sub-space of concepts where the best solutions are not present, as illustrated in Figure 3.3 where a search may become trapped in sub-optimal *local maxima*.

A basic beam search algorithm by downward refinement which maintains a set of best hypotheses relative to a utility function **u** is shown as Algorithm 3. This algorithm also incorporates an anti-monotonic quality function $\mathcal{Q}$ to determine if a refined hypothesis should be added to the frontier, or pruned. Note that while accuracy (Definition 3.3.3) can be used to rank hypotheses as a utility function, it is neither monotonic nor anti-monotonic, so cannot be used for pruning hypotheses from a search, unlike relative frequency.

One method of mitigating the risk that Algorithm 3 becomes trapped in a search space around a local maxima is to introduce randomness in the search. One such method for achieving this is known as *stochastic beam search*. In stochastic beam search, the reinitialisation of the next beam (Lines 14 to 19) is modified to select candidates at random with a probability which is proportional to a function of their

Algorithm 3 A basic *best-first beam search* with downward refinement operator ρ to search the space of concepts $(\mathcal{L}, \sqsubseteq)$ relative to examples $\mathcal{E}$, where $\mathbf{u}$ is a utility function ranking better hypotheses with larger values, and where $\mathcal{Q}$ is an anti-monotonic quality function assessing if hypotheses can be considered solutions. The maximum beam width is denoted by b_{max}.

```
1:  $B := \{\top\}$                              ▷ The hypothesis frontier beam of search candidates
2:  $S := \emptyset$                                                   ▷ The set of solutions
3:  while $|B| > 0$ do                          ▷ While the frontier beam is non-empty
4:      $E = \emptyset$                                        ▷ Initialise the expansion set
5:      for all $C \in B$ do
6:          for all $D \in \rho(C)$ do
7:              if $\mathcal{Q}(D, \mathcal{E}) = true$ then         ▷ Hypothesis $D$ is a sufficient candidate
8:                  $S := S \cup \{D\}$                                    ▷ Capture solution $D$
9:              else
10:                 $E := E \cup \{D\}$                       ▷ Include $D$ in the expansion set
11:             end if
12:         end for
13:     end for
14:     $B := \emptyset$                                              ▷ Reinitialise the beam
15:     while $|E| > 0$ and $|B| < b_{max}$ do
16:         $D \in \arg\max_{D \in E} \mathbf{u}(D)$                        ▷ Arbitrary best refinement
17:         $E := E \setminus \{D\}$
18:         $B := B \cup \{D\}$                                  ▷ Include $D$ in the next beam
19:     end while
20: end while
```

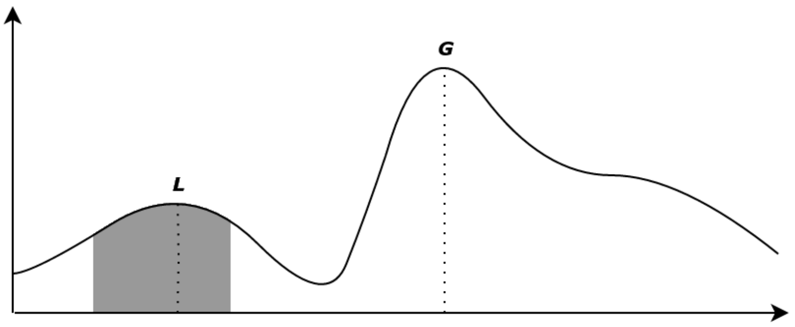


Figure 3.3: A graph where the curve represents the space of all hypotheses (horizontal axis) against their performance (vertical axis). An algorithm (such as a beam search) which limits the search to the shaded region may only find hypotheses at *local maximum L* as being best, and will fail to locate the best solution(s) at the *global maximum G*.

utility. A common method is to use the Gibbs distribution $e^{\frac{-\mathbf{c}(D)}{T}}$ for a concept D and some value $T \in \mathbb{R}$ where the $\mathbf{c} : \mathcal{L} \mapsto \mathbb{R}$ is a *cost* function, and may be based on a utility function $\mathbf{u}$. This distribution reflects the intuition that stronger hypotheses should be selected with greater probability than weaker ones. *Stochastic refinement* is another approach which incorporates such random selection directly into the behaviour of a refinement operator which refines to new candidates with certain probabilities [98].

3.4.4 The Limitations of Open-World DL Learning

Up until now, we have not defined precisely how to compute the cover of a concept C relative to a set of examples $\mathcal{E}$. One such method called *learning from entailment* is to perform *instance retrieval* for any concept C generated in a search.

Definition 3.4.13. ***(Learning from Entailment)*** *Given a hypothesis C and example $e \in \mathcal{E}$, **learning from entailment** in a DL knowledge base $\mathcal{K}$ describes the setting where $covers(C, e)$ iff $\mathcal{K} \models C(e)$.*

Definition 3.4.13 is due to De Raedt [23], but is adapted here for use in DL knowledge-bases. With respect to learning in DL knowledge bases, learning from entailment poses significant practical and theoretical limitations on the search. From a practical standpoint, instance retrieval can be a highly computationally expensive operation. For any DL knowledge $\mathcal{K}$ of high expressivity such as $\mathcal{SROIQ}$, instance checking is at least exponential in the size of the TBox and ABox. Recall that our goal is to develop methods of machine learning and data mining in DL knowledge bases which employ the learning as search algorithms of Section 3.4.3. These algorithms are designed to generate-and-test many new concepts, and rely on the computation of hypothesis coverage to be fast. Unfortunately, to perform instance checking, new concepts generated by refinement which were previously unknown to a knowledge base need to be re-classified with respect to the TBox. Depending on the size of the TBox and ABox of the knowledge base, this can be an expensive step which is at odds with our goal of searching a space of concepts efficiently. For example, re-classification of a $\mathcal{SROIQ}$ knowledge base with logical reasoning has N2ExpTime complexity [46]. When contrasted with the complexity of querying in most other machine learning and data mining settings which employ, for example, a relational database to test for the coverage of hypotheses with low complexity (PTime, or even LogSpace), this complexity is unacceptable for the sake of the performance of a learning algorithm.

Secondly, learning from entailment in DL knowledge bases is adversely affected by the open-world assumption (OWA). For example, consider the hypothesis expression $\forall r.C$, along with instances $r(e, y)$ and $C(y)$ asserted to the ABox. Even with an empty TBox, a knowledge base may not entail $\mathcal{K} \models (\forall r.C)(e)$ as, according to the OWA, we do not know if there are other r-successors of e which are not instances of C. Therefore, as other interpretations exist which permit r-successors of e which are not instances of C, e will never be attributed as an instance of this hypothesis by entailment. Similarly, the same can be said for expressions such as ${\leqslant}^{n} r.C$. Concept negation such as $\neg C$ is also affected for the same reason, as unless an example e has been explicitly asserted to be an instance of an expression $E \equiv \neg C$, a knowledge base may not entail $\mathcal{K} \models (\neg C)(e)$ as this also remains unknown. These restrictions pose limitations on the expressiveness of a hypothesis language such as $\mathcal{SROIQ}$ concepts, which we otherwise desire as such a language permits powerful descriptions of patterns in a knowledge base for learning.

Typically in machine learning and data mining, it is assumed that examples are completely specified in that there is are missing data assertions to describe them. This assumption is at odds with the OWA made by DL knowledge bases, where instances are assumed to be incompletely specified. Therefore, it is reasonable to want to make a closed world assumption (CWA) over a DL knowledge base which permits the default assumption to reflect our intention that examples are completely specified. However, because the CWA is highly restrictive by imposing many more assumptions, the number of possible models of any knowledge base under the CWA diminishes, thus increasing the possibility of inconsistency within knowledge bases with TBoxes based on expressive logics such as $\mathcal{SROIQ}$.

In the next section, we will define an appropriate interpretation for $\mathcal{SROIQ}$ concepts which reflects the CWA, and demonstrate how such an interpretation can be used for efficient coverage checking of concepts. Such an interpretation effectively addresses both our concerns around the use of the OWA and expensive knowledge base entailment. We then discuss the implications of the use of the interpretation in the context of learning relative to complex background knowledge in DL knowledge bases.

3.5 Closed-World DL Learning

In Section 3.4.4, we discussed the limitations of *learning from entailment* in DL knowledge bases which make the OWA. In this section, we describe how we address these limitations with a different setting for learning known as *learning from interpretations*, which is also adapted from De Raedt [23].

Definition 3.5.1. ***(Learning from Interpretations)*** *Given a hypothesis C and example $e \in \mathcal{E}$,* ***learning from interpretations*** *in a DL knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ describes the setting where covers(C, e) iff there exists some interpretation $\mathcal{J}$ where $\mathcal{J} \models C(e)$, and where cover$(C, \mathcal{E}) = \{e' \in \mathcal{E} \mid \mathcal{J} \models C(e')\}$ and $\mathcal{J} \models \mathcal{A}$.*

In this setting, we will be describing the use of a *single* interpretation $\mathcal{J}$ which is at least a model of $\mathcal{A}$. In practical terms, this means that $\mathcal{J}$ is a *fixed model* which reflects the set of examples as all of the asserted instance data in a knowledge base. As we will discuss in Section 3.5.2, it may be unlikely that a DL knowledge base with expressive TBox axioms has a single interpretation. This means that $\mathcal{J}$ may not model all background knowledge, $\mathcal{J} \not\models \mathcal{T}$, however we may still utilise $\mathcal{J}$ for learning concepts efficiently. We will now describe an instance of such an interpretation and then discuss its advantages and disadvantages over the typical open-world interpretation $\mathcal{I}$ (Definition 3.2.5) for learning DL concepts.

3.5.1 A Closed-World Interpretation for $\mathcal{SROIQ}$ Concepts

An example of an interpretation applicable for interpreting $\mathcal{SROIQ}$ concepts which reflects the CWA has been described by Tao et. al. [100] as the *IC-interpretation* for handling so-called *integrity constraints* in OWL2-DL.

Definition 3.5.2. ***(IC Interpretation)*** *The* ***IC interpretation*** *[100] is defined as the pair $(\mathcal{I},\mathcal{U}) = (\Delta^{(\mathcal{I},\mathcal{U})}, \cdot^{(\mathcal{I},\mathcal{U})})$ over the signature (N_C, N_R, N_I) of a knowledge base $\mathcal{K}$ (Definition 3.2.2) where $\mathcal{I}$ is a $\mathcal{SROIQ}$ interpretation (Definition 3.2.5) $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, $\mathcal{U}$ is the set of all interpretations $\mathcal{I}$ of $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$, $\Delta^{(\mathcal{I},\mathcal{U})}$ is a non-empty set, a is an individual $a \in N_I$, A is a concept name $A \in N_C$, r is a role name $r \in N_R$, and where $\cdot^{(\mathcal{I},\mathcal{U})}$ maps:*

$$A^{(\mathcal{I},\mathcal{U})} = \{x^{\mathcal{I}} \mid x \in N_I \text{ s.t. } \forall \mathcal{J} \in \mathcal{U}, x^{\mathcal{J}} \in A^{\mathcal{J}}\}$$
$$r^{(\mathcal{I},\mathcal{U})} = \{\langle x^{\mathcal{I}}, y^{\mathcal{I}}\rangle \mid x, y \in N_I \text{ s.t. } \forall \mathcal{J} \in \mathcal{U}, \langle x^{\mathcal{J}}, y^{\mathcal{J}}\rangle \in r^{\mathcal{J}}\}$$
$$a^{(\mathcal{I},\mathcal{U})} = a^{\mathcal{I}}$$

and which extends to arbitrary concepts inductively as follows, where $\sharp$ denotes set cardinality:

$$\begin{array}{rcl}
\top^{(\mathcal{I},\mathcal{U})} & = & \Delta^{(\mathcal{I},\mathcal{U})} \\
\bot^{(\mathcal{I},\mathcal{U})} & = & \emptyset \\
(\neg C)^{(\mathcal{I},\mathcal{U})} & = & \Delta^{(\mathcal{I},\mathcal{U})} \setminus C^{(\mathcal{I},\mathcal{U})} \\
(C \sqcap D)^{(\mathcal{I},\mathcal{U})} & = & C^{(\mathcal{I},\mathcal{U})} \cap D^{(\mathcal{I},\mathcal{U})} \\
(C \sqcup D)^{(\mathcal{I},\mathcal{U})} & = & C^{(\mathcal{I},\mathcal{U})} \cup D^{(\mathcal{I},\mathcal{U})} \\
\{x\}^{(\mathcal{I},\mathcal{U})} & = & \{x : x \in \Delta^{(\mathcal{I},\mathcal{U})}\} \\
(\exists r.C)^{(\mathcal{I},\mathcal{U})} & = & \{x : x \in \Delta^{(\mathcal{I},\mathcal{U})} \text{ s.t. } \exists y.\langle x, y\rangle \in r^{(\mathcal{I},\mathcal{U})} \wedge y \in C^{(\mathcal{I},\mathcal{U})}\} \\
(\forall r.C)^{(\mathcal{I},\mathcal{U})} & = & \{x : x \in \Delta^{(\mathcal{I},\mathcal{U})} \text{ s.t. } \forall y.\langle x, y\rangle \in r^{(\mathcal{I},\mathcal{U})} \rightarrow y \in C^{(\mathcal{I},\mathcal{U})}\} \\
({\geqslant}n\, r.C)^{(\mathcal{I},\mathcal{U})} & = & \{x : x \in \Delta^{(\mathcal{I},\mathcal{U})} \text{ s.t. } \sharp\{y.\langle x, y\rangle \in r^{(\mathcal{I},\mathcal{U})} \wedge y \in C^{(\mathcal{I},\mathcal{U})}\} \geqslant n\} \\
({\leqslant}n\, r.C)^{(\mathcal{I},\mathcal{U})} & = & \{x : x \in \Delta^{(\mathcal{I},\mathcal{U})} \text{ s.t. } \sharp\{y.\langle x, y\rangle \in r^{(\mathcal{I},\mathcal{U})} \wedge y \in C^{(\mathcal{I},\mathcal{U})}\} \leqslant n\}
\end{array}$$

In this way, the IC interpretation of atomic concepts $A^{(\mathcal{I},\mathcal{U})}$ and roles $r^{(\mathcal{I},\mathcal{U})}$ reflects what is *known* in $\mathcal{K}$, or in other words, the sets and tuples comprised of known individuals $a^{(\mathcal{I},\mathcal{U})}$ which are entailed as instances of each (following from Definition 3.2.18). Furthermore, this interpretation is defined to make the *weak unique name assumption*, in that it is intended for use under a weaker form of the unique name assumption (Definition 3.2.9).

Definition 3.5.3. ***(Weak Unique Name Assumption)*** *Given a knowledge base $\mathcal{K}$, the set of all first-order models $\mathcal{U}$ of $\mathcal{K}$, the IC interpretation $(\mathcal{I},\mathcal{U})$ and two named individuals $i, j \in N_I$ with distinct names $i \neq j$, the* ***weak unique name assumption (weak UNA)*** *describes the case where, if $\forall \mathcal{I} \in \mathcal{U} : i^{\mathcal{I}} = j^{\mathcal{I}}$, then $i^{(\mathcal{I},\mathcal{U})} = j^{(\mathcal{I},\mathcal{U})}$, otherwise $i^{(\mathcal{I},\mathcal{U})} \neq j^{(\mathcal{I},\mathcal{U})}$.*

Given that all instances in $\Delta^{(\mathcal{I},\mathcal{U})}$ $(\Delta^{(\mathcal{I},\mathcal{U})} \times \Delta^{(\mathcal{I},\mathcal{U})})$ which are not also instances of $C^{(\mathcal{I},\mathcal{U})}$ $(r^{(\mathcal{I},\mathcal{U})})$ are assumed to lie in the negation $(\neg C)^{(\mathcal{I},\mathcal{U})}$ $((\neg r)^{(\mathcal{I},\mathcal{U})})$, this reflects the CWA as we desire for DL learning, and furthermore, the weak UNA permits us to respect any entailments which a knowledge base makes about the equivalence of individuals, and makes the unique name assumption otherwise. In terms of efficiency, the interpretation of arbitrary complex concepts C may be performed directly in terms of $(\mathcal{I},\mathcal{U})$, which permits us to perform coverage checking for any concept C by way of checking, for any individual e, if the interpretation $(\mathcal{I},\mathcal{U}) \models C(e)$. Given that $(\mathcal{I},\mathcal{U})$ is fixed, $(\mathcal{I},\mathcal{U}) \models C(e)$ is at most an ExpTime operation in the size of $(\mathcal{I},\mathcal{U})$ and the concept C (§5.2.1) which contrasts favourably with the typically more expensive $\mathcal{K} \models C(e)$ under the OWA for expressive DLs as discussed in Section 3.4.4. For such coverage checking to proceed for any arbitrary complex concept expression C relative to a given $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$, the interpretation $(\mathcal{I},\mathcal{U})$ must first be *materialised* over all named individuals, concept and roles. This necessarily requires the full classification of $\mathcal{K}$ with logical reasoning under the open-world interpretation $\mathcal{I}$ which, as we discussed in Section 3.4.4, is computationally expensive. However, this step is only required *once*, as new arbitrary concept expressions may then be interpreted directly with $(\mathcal{I},\mathcal{U})$ over the atomic concepts, roles and individuals of which they are comprised. In this way, $(\mathcal{I},\mathcal{U})$ reflects a *database* (which is effectively a finite model with closed world semantics), and arbitrary complex concepts can be thought of as *queries* to that database. This compares favourably to the alternative of learning from entailment, where new concept expressions must be integrated into $\mathcal{K}$ and re-classified using logical reasoning each time, otherwise presenting an impediment to efficient learning by generate-and-test methods.

While we have so far argued that learning from interpretations is an appropriate setting for machine learning and data mining over DL knowledge bases as it reflects the CWA and permits for efficient hypothesis coverage checking, there are certain limitations to learning from interpretation $(\mathcal{I},\mathcal{U})$ such as the fact it may not model a knowledge base with incomplete data or expressive TBox axioms. We now discuss these limitations in the next section.

3.5.2 Limitations of Learning from Interpretations in DL Knowledge Bases

Under the closed-world interpretation $(\mathcal{I},\mathcal{U})$, it is possible to induce a concept expression C which has a non-empty interpretation $C^{\mathcal{I}\mathcal{U}} \neq \emptyset$ but where C is actually unsatisfiable ($C^{\mathcal{I}} = \emptyset$) under the standard interpretation $\mathcal{I}$ with an open-world assumption.

Example 3.5.4. *Consider a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{A} = \{A(i), r(i,j), C(j)\}$ and $\mathcal{T} = \{A \sqsubseteq {\geqslant}2 r.C\}$, describing how instances of A necessarily have two r-successors which are instances of C. Recall that an interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ if, for all axioms $C \sqsubseteq D \in \mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (Definition 3.2.10). In this example, $A^{(\mathcal{I},\mathcal{U})} = \{i\}$ and $({\geqslant}2 r.C)^{(\mathcal{I},\mathcal{U})} = \emptyset$ and therefore, $(\mathcal{I},\mathcal{U}) \not\models \mathcal{T}$. This is because the interpretation $({\geqslant}2 r.C)^{(\mathcal{I},\mathcal{U})}$ is composed exclusively of known data in $\mathcal{K}$, and only one r-successor of i was asserted and therefore known. If information is missing from $\mathcal{A}$ with respect to $\mathcal{T}$, such as in the case that instances of A are expected to have at least two r-successors yet only one was asserted, it may lead a learning algorithm relying on $(\mathcal{I},\mathcal{U})$ for computing hypothesis coverage to produce an expression which is unsatisfiable with respect to $\mathcal{T}$. In this case, one such expression is $(A \sqcap {\leqslant}1 r.C)$ where $(A \sqcap {\leqslant}1 r.C)^{(\mathcal{I},\mathcal{U})} = \{i\}$, however this expression is clearly unsatisfiable with respect to $\mathcal{T}$ given it contains the contradictory axiom $(A \sqsubseteq {\geqslant}2 r.C)$.*

Example 3.5.4 highlights the motion of data which is *missing* in $\mathcal{A}$ with respect to $\mathcal{T}$, in that if it is expected that examples are structured in a certain way by the definition of axioms in $\mathcal{T}$, then data which conforms to this structure must have been asserted to $\mathcal{A}$. Otherwise, a closed-world interpretation such as $(\mathcal{I},\mathcal{U})$ may not model $\mathcal{T}$. Similarly, axioms may exist in $\mathcal{T}$ which ambiguously define concepts and this has implications for their interpretation in a closed-world, as shown in Example 3.5.5.

Example 3.5.5. *Consider a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, concept names A, B, C, and where $(A \sqsubseteq B \sqcup C) \in \mathcal{T}$ and $A(x) \in \mathcal{A}$. Assume that $\mathcal{K} \not\models B(x)$ and $\mathcal{K} \not\models C(x)$, so that under the closed-world interpretation $(\mathcal{I},\mathcal{U})$, we have $B^{(\mathcal{I},\mathcal{U})} = \emptyset$ and $C^{(\mathcal{I},\mathcal{U})} = \emptyset$. However, even though $\mathcal{K} \models (B \sqcup C)(x)$, we have $(\mathcal{I},\mathcal{U}) \not\models (B \sqcup C)(x)$ because it is unknown whether x is an instance of B, C, or both. While the open-world interpretation $\mathcal{I}$ admits several possible models of $\mathcal{K}$, the single closed-world interpretation $(\mathcal{I},\mathcal{U})$ is not a model of $\mathcal{K}$.*

Furthermore, the use of a closed-world interpretation such as $(\mathcal{I},\mathcal{U})$ for determining the coverage of concepts as hypotheses in a search may lead to erroneous characterisations of performance as shown in Example 3.5.6.

Example 3.5.6. *Consider a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{A} = \{E_1(i), E_2(j), r(i,k)\}$ and $\mathcal{T} = \{E_1 \sqsubseteq C, E_2 \sqsubseteq C, C \sqsubseteq \exists r.\top\}$ for the labelled example classes E_1 and E_2. In this case, we find that $(\exists r.\top)^{\mathcal{I}} = \{i, j, \ldots\}$, however $(\exists r.\top)^{(\mathcal{I},\mathcal{U})} = \{i\}$ as no r-successor of j is known in $\mathcal{A}$. From the perspective of a machine learner performing binary classification, the cover of hypothesis $\exists r.\top$ computed by $(\mathcal{I},\mathcal{U})$ appears to correctly describe all instances of example class E_1 and no instances of E_2, and is thus considered a perfect characterisation* *of E_1. However, clearly such a characterisation is incorrect as at least one r-successor of j is implied by $\mathcal{T}$, yet it simply remains unknown under the OWA.*

The underlying problem in each of the Examples 3.5.4, 3.5.5 and 3.5.6 relates to data which is *missing in $\mathcal{A}$ with respect to $\mathcal{T}$*. In each of these examples, the TBox implied the existence of data or knowledge which was not asserted to the ABox, such as missing role tuples or class instance assertions. In order to make use of $(\mathcal{I},\mathcal{U})$ for learning in light of these limitations, we can use it for the purpose it was originally intended, which is to perform *integrity checking* of assertions in $\mathcal{A}$ against axioms in $\mathcal{T}$ [100].

Definition 3.5.7. ***(Integrity Checking)*** *Given a $\mathcal{SROIQ}$ knowledge base $\mathcal{K} = (\mathcal{T},\mathcal{A})$, the task of* ***integrity checking*** *seeks to test, for all axioms $C \sqsubseteq D \in \mathcal{T}$, if $C^{(\mathcal{I},\mathcal{U})} \subseteq D^{(\mathcal{I},\mathcal{U})}$ holds. For an axiom $C \sqsubseteq D \in \mathcal{T}$, any individual $i \in C^{(\mathcal{I},\mathcal{U})}$ where $i \notin D^{(\mathcal{I},\mathcal{U})}$ fails the integrity check against $C \sqsubseteq D$.*

Integrity checking in $\mathcal{SROIQ}$ with the closed-world interpretation $(\mathcal{I},\mathcal{U})$ can be used to compute which data asserted to an ABox is known to be incomplete relative to axioms in the TBox, subject to certain restrictions on the types of expressions in the latter [100]. In the next section, we will discuss how integrity checking can be used to assess the suitability of data in the ABox of a knowledge base against axioms in a TBox.

3.5.2.1 Detecting and Handling Incomplete Data

Data in the ABox of a knowledge base which is incomplete with respect to the TBox poses challenges to learning algorithms, as we have shown. Generally, a learning algorithm which generates concept expressions C and assesses their performance based on their cover with the IC interpretation is influenced by the *distribution* of individuals and literals in the knowledge base relative to concept terms and sub-expressions of which C is composed. In general, it is difficult to characterise *which* missing data will influence a learner to generate problematic concepts, as this depends on the particular distribution of missing data together with the particular learning strategy being used. However, in order to minimise the possibility of encountering problems described in Section 3.5.2, we may employ the use of an algorithm for integrity constraint checking under the IC interpretation to determine which individual data fails integrity checks over particular TBox axioms, such as that described by Tao et. al. [100], as was the original intention behind the development of the IC interpretation.

For example, consider the case where we have knowledge base $\mathcal{K} = (\mathcal{A}, \mathcal{T})$, concepts C, D where $(C \sqsubseteq D) \in \mathcal{T}$ and individual i where $C(i) \in \mathcal{A}$. While individual $i^{(\mathcal{I},\mathcal{U})} \in C^{(\mathcal{I},\mathcal{U})}$, we may have $i^{(\mathcal{I},\mathcal{U})} \notin D^{(\mathcal{I},\mathcal{U})}$. Given individual i was implicated in the failure of an integrity check on the axiom $(C \sqsubseteq D)$, we have several options.

Firstly, we may wish to manually repair examples which incorporate individuals such as i by adding sufficient extra data to the ABox so as to satisfy the TBox axioms which were implicated in the failure of integrity checks. An example e which may be represented by an RDF graph and mapped to a particular set of ABox assertions may contain or refer to individual i. As we know that i is incompletely defined, we can highlight it for attention so it can be corrected. This requires a user to understand precisely how to add new instance data linked to i which satisfies the violated axioms, which may be an unreasonable assumption as axioms in highly expressive knowledge bases such as those based on OWL2-DL may be quite complex. Notably, this is a research problem in itself, and has been addressed with methods and algorithms to automatically generate missing data [99].

Alternatively, a simpler approach is to exclude from a learning problem all examples e which contain or refer to individuals which were implicated in the failure of integrity checks. Completely excluding examples from a learning problem may be feasible if an insignificant proportion of examples are to be removed, however the approach becomes prohibitively restrictive if it requires the exclusion of too many examples which may result in an insufficient training and test set for a learning problem.

Lastly, another approach is to prevent a learning algorithm from inducing concepts which cover any individuals or literals which fail integrity checks. In this way, a learner will not attempt to describe any data which may be incompletely defined. This approach prevents a learner from generating unreliable expressions which are posed over incomplete data, effectively excluding any features they represent from the learning problem. While this approach may also be overly restrictive if there are many instances which violate integrity checks in many contexts, it is straightforward to restrict a learning algorithm to avoid the induction of concepts, or subexpressions thereof, which cover such incompletely defined data.

In the remainder of this thesis, we will make the simplifying assumption that any IC interpretation $(\mathcal{I},\mathcal{U})$ does not lead to integrity violations relative to the TBoxes for the knowledge bases we consider. This is not an unreasonable assumption, as often TBoxes may still model significant amounts of background knowledge using axioms for which instance data does not violate integrity checks. TBoxes which only contain simple inclusion axioms such as a taxonomy of named concepts, or at most contains inclusion axioms defining concepts as subclasses of simple existential role restrictions such as with the DL $\mathcal{EL}$, will often not pose a challenge in this regard.

We also believe that the benefits of using a closed-world interpretation such as $(\mathcal{I},\mathcal{U})$ for efficient concept refinement and coverage testing in learning problems far outweigh the drawbacks which include the possibility of generating expressions which violate axioms in $\mathcal{T}$. This is particularly the case when we aim to use concept induction to perform analyses over the data such as data mining for patterns, where concepts are induced to be human readable descriptions of interesting patterns, and are not intended to be re-incorporated into the TBox, which is otherwise known as *class learning*. Our approach is distinct from the goal of much of the related research in DL learning which does focus on class learning with the intention of integrating concepts back into a TBox.

3.5.2.2 Independence of the expressivity of induced concepts to $\mathcal{K}$

The language expressivity of concepts induced by a learning algorithm can be independent of the expressivity of the concepts which pre-exist in the knowledge base. For example, $\mathcal{T}$ may contain complex axioms with concepts composed with the expressivity of $\mathcal{SROIQ}$, however hypotheses can be confined to concepts which are composed of logical constructs only found in $\mathcal{EL}$. Conversely, the expressivity of $\mathcal{K}$ may be minimal, such as when $\mathcal{T}$ is empty (without background knowledge) and $\mathcal{A}$ assertions are made against atomic concepts. Hypotheses induced over such a knowledge base can still be very complex, such as concepts induced with the language $\mathcal{ALCOQ}(\mathcal{D})$ to capture interesting groups of individuals.

The expressivity of the language for hypotheses can be selected to suit the particular problem being solved. For example, if qualified cardinality restrictions and disjunction are not desired in the composition of hypotheses for a given problem, they can be omitted from the learning process by excluding their use by a refinement operator as a form of *language bias*. This may also be chosen for performance reasons as such constructs inflate the search space of concepts for induction. Similarly, if assertions in a knowledge base contain concrete domain elements such as numbers, strings and boolean values, we may wish to employ hypotheses which permit restrictions on these values even if the TBox makes no mention of these. This may particularly be the case if we are dealing with knowledge bases which capture numerical experimental results in which we are especially interested.

3.6 Summary

In this chapter, we have described the relationship between RDF data described with OWL and the underlying Description Logics which underpin OWL, along with basic notions of how DL concept induction for classification and subgroup discovery can be performed with refinement-based search. We then highlighted the deficiencies of learning under an open-world assumption and formalised a suitable setting for learning under a closed-world setting which addresses these concerns. From here, the contributions of this thesis will focus on how the closed-world learning setting can be leveraged to deliver performant DL learning in systems which employ refinement operators especially for classification and subgroup discovery.

Chapter 4

Concept Induction by Refinement Operators

In the previous chapter, we described both the prerequisites for concept induction and various learning settings for learning over DL knowledge bases, particularly with high expressivity where the DL corresponds to OWL2-DL. Our goal is to be able to apply such techniques to large DL knowledge bases to permit efficient learning to support classification (machine learning) and subgroup discovery (data mining).

State-of-the-art learning systems such as DL-Learner [58] employ a refinement procedure in learning which is not optimised for large knowledge bases. The refinement operator used in this system was one of the first to be posed over the highly expressive DL known as $\mathcal{SROIQ}(D)$ for describing OWL2-DL classes, and as such was integrated into DL-Learner as a proof-of-concept. While certain optimisations were identified and implemented in DL-Learner, we significantly improve on the performance of the system through our understanding of concept induction under a closed-world assumption and in the setting of learning from interpretations, as presented in the last chapter. The main contributions made in this chapter are:

- Identification of the inefficiencies of the refinement operator used in the state-of-the-art system DL-Learner for learning in $\mathcal{SROIQ}(D)$ knowledge bases (§4.1);
- Introduction of a method that structures a closed-world interpretation to reveal knowledge about the structure of the concept search space (§4.2);
- Definition of a modified refinement operator in terms of knowledge gained from a structured closed-world interpretation which addresses the inefficiencies of the previously defined operator (§4.3).

We begin with a discussion on the state-of-the-art refinement operator used in the DL-Learner system and describe various limitations which we will address in subsequent sections, before assembling a new refinement operator which addresses these limitations. In the next chapter, we describe how the new refinement operator can be incorporated into supervised learning algorithms for classification and subgroup discovery for efficient learning in $\mathcal{SROIQ}(D)$ knowledge-bases.

4.1 A Refinement Operator for OWL2-DL Classes

In this section we describe the downward refinement operator ρ used by the state-of-the-art system DL-Learner which is used for concept induction for learning OWL2-DL classes based on highly expressive DL $\mathcal{SROIQ}(D)$ [58]. This downward refinement operator was developed from a careful analysis of the properties of refinement operators for DLs [56] which recognised that while it is complete, it is also redundant, and not proper. Nevertheless, it is used for effective concept induction in DL-Learner and has been successfully applied to various concept learning problems. The definition of the refinement operator ρ relies on several functions, such as $sh_\downarrow$ ($sh_\uparrow$) which traverses the subsumption hierarchy of concept names and role names in $\mathcal{T}$ as follows:

$$\begin{aligned} sh_\downarrow(A) &= \{A' \mid A' \in N_C, A' \sqsubset A, \neg\exists A'' \in N_C \text{ s.t. } A' \sqsubset A'' \wedge A'' \sqsubset A\} \\ sh_\downarrow(r) &= \{r' \mid r' \in N_R, r' \sqsubset r, \neg\exists r'' \in N_R \text{ s.t. } r' \sqsubset r'' \wedge r'' \sqsubset r\} \end{aligned}$$

The functions for upward traversal of the subsumption hierarchy $sh_\uparrow(A)$ and $sh_\uparrow(r)$ are defined similarly. The operator also relies on the functions $ar(r)$ ($ad(r)$) which map role names in N_R to atomic concept names in N_C which cover all instances in the range (domain) of a role r, so as to restrict concept refinements in the range (domain) of r to appropriate concept names.

The set of concepts which are available for refinement in the context of the range of a quantified role expression is described as the set M_B as follows:

$$\begin{aligned} M_B \;=\; & \{A \mid A \in N_C, A \sqcap B \not\equiv \bot, A \sqcap B \not\equiv B, \neg\exists A' \text{ s.t. } A' \in N_C \text{ and } A \sqsubset A'\} \cup \\ & \{\neg A \mid A \in N_C, \neg A \sqcap B \not\equiv \bot, \neg A \sqcap B \not\equiv B, \neg\exists A' \text{ s.t. } A' \in N_C \text{ and } A' \sqsubset A\} \cup \\ & \{\exists r.(\top) \mid r \in mgr_B\} \cup \\ & \{\forall r.(\top) \mid r \in mgr_B\} \end{aligned}$$

where mgr_B is the set of most general roles applicable to individuals in the domain of B defined as

$$mgr_B = \{r \mid r \in N_R, ad(r) \sqcap B \not\equiv \bot, \neg\exists r' \in N_R \text{ s.t. } r \sqsubset r' \wedge ad(r') \sqcap B \not\equiv \bot\}.$$

The refinement operator ρ_B defined in terms of the context B, where B is the atomic named concept describing the range of a role r, as follows:

$$\rho_B(C) = \begin{cases} \emptyset & \text{if } C = \bot \\ \{C_1 \sqcup \ldots \sqcup C_n | C_i \in M_B (1 \leq i \leq n\} & \text{if } C = \top \\ \{A' \mid A' \in sh_{\downarrow}(A)\} & \text{if } C = A \ (A \in N_C) \\ \quad \cup \{A \sqcap D | D \in \rho_B(\top)\} & \\ \{\neg A' \mid A' \in sh_{\uparrow}(A)\} & \text{if } C = \neg A \ (A \in N_C) \\ \quad \cup \{\neg A \sqcap D | D \in \rho_B(\top)\} & \\ \{\exists r.E | A = ar(r), E \in \rho_A(D)\} & \text{if } C = \exists r.D \\ \quad \cup \{\exists r.(D) \sqcap E | E \in \rho_B(\top)\} & \\ \quad \cup \{\exists s.(D) | s \in sh_{\downarrow}(r)\} & \\ \{\forall r.E | A = ar(r), E \in \rho_A(D)\} & \text{if } C = \forall r.D \\ \quad \cup \{\forall r.(D) \sqcap E | E \in \rho_B(\top)\} & \\ \quad \cup \{\forall s.D | s \in sh_{\downarrow}(r)\} & \\ \{C_1 \sqcap \ldots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \ldots \sqcap C_n | & \text{if } C = C_1 \sqcap \ldots \sqcap C_n \ (n \geq 2) \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} & \\ \{C_1 \sqcup \ldots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \ldots \sqcup C_n | & \text{if } C = C_1 \sqcup \ldots \sqcup C_n \ (n \geq 2) \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} & \\ \quad \cup \{(C_1 \sqcup \ldots \sqcup C_n) \sqcap D | D \in \rho_B(\top)\} & \end{cases}$$

The downward refinement operator ρ is then defined in terms of ρ_B as

$$\rho(C) = \begin{cases} \{\bot\} \cup \rho_\top(C) & \text{if } C = \top \\ \rho_\top(C) & \text{otherwise.} \end{cases}$$

Example 4.1.1 demonstrates the application of ρ from $\top$ by illustrating a sequence of single-step refinements of various subexpressions over concepts describing the composition of chemical molecules.

Example 4.1.1. *Starting from the top concept $\top$, the downward refinement operator ρ can be applied in a number of ways to generate more complex concepts in a step-wise manner, such as the following refinement chain describing chemical compounds:*

$$\begin{aligned}
\top &\rightsquigarrow Compound\\
&\rightsquigarrow Compound \sqcap \exists hasPart.(\top)\\
&\rightsquigarrow Compound \sqcap \exists hasPart.(\top) \sqcap \forall hasPart.(\top)\\
&\rightsquigarrow Compound \sqcap \exists hasPart.(Ion) \sqcap \forall hasPart.(\top)\\
&\rightsquigarrow Compound \sqcap \exists hasPart.(Ion) \sqcap \forall hasPart.(\neg Metal)\\
&\rightsquigarrow Compound \sqcap \exists hasPart.(Ion \sqcap \exists hasGroup.(\top)) \sqcap \forall hasPart.(\neg Metal)\\
&\rightsquigarrow Compound \sqcap \exists hasPart.(Ion \sqcap \exists hasGroup.(Carboxyl)) \sqcap \forall hasPart.(\neg Metal)
\end{aligned}$$

This example describes a single refinement chain amongst potentially very many in the space of possible refinements to other expressions, and demonstrates how the refinement operator can be applied in various ways within each expression.

Following on from Example 4.1.1, we observe how the downward operator ρ may refine to concepts which cannot describe any examples, as shown in Example 4.1.2.

Example 4.1.2. *Consider the following complex concept C describing chemical compounds where:*

$$C = \overbrace{Compound}^{1} \sqcap \exists hasPart.(\overbrace{Ion}^{2} \sqcap \exists hasGroup.(Carboxyl)) \sqcap \forall hasPart.(\neg Metal)$$

The concept C describes the set of chemical compounds with no part metal and at least one part ion consisting of at least one carboxyl group. The downward operator ρ may permit the following refinements of labelled subexpressions 1,2 which render C as unsatisfiable, as follows:

No.	*Refinement*	*Observations*
1.	*Compound* $\rightsquigarrow ZnCl_2$	*Zinc chloride $ZnCl_2$ has part metal (zinc), and neither ion has part carboxyl group.*
2.	*Ion* $\rightsquigarrow Ion \sqcap NH_4^+$	*Ammonium NH_4^+ is a subclass of Ion, but has no carboxyl groups.*

In this example, refinements which were permissible to subexpressions of C in isolation produced the expressions $ZnCl_2 \sqcap \forall hasPart.(\neg Metal)$ and $NH_4^+ \sqcap \exists hasGroup.(Carboxyl)$ which are both unsatisfiable. These refinements were permitted on the basis of the axioms captured in the TBox, such as the fact that $ZnCl_2 \sqsubseteq Compond$ and $NH_4^+ \sqsubseteq Ion$.

Example 4.1.2 highlights an inefficiency with the operator ρ where refinements of subexpressions of a concept can be generated in isolation to the rest of the expression. In this way, refinements may be generated which are of potentially no value, such as when they are unsatisfiable. Consider the case where we have a knowledge base consisting of a large number of examples as assertions to the ABox, and where the TBox contains many thousands of subclasses of $Compound$ which have no carboxyl groups, or have no part metal. In such a case, a learning algorithm which uses ρ to generate refinements may evaluate very many ultimately unsatisfiable concepts over the large number of examples, which may otherwise waste computational resources.

We may be tempted to address such inefficiency by relying on a DL reasoner to determine if any concept generated by a refinement operator is suitable. For example, if in Example 4.1.2 the TBox contained the axioms $ZnCl_2 \sqsubseteq \exists hasPart.(Zn), Zn \sqsubseteq Metal$, then the unsatisfiability in case (1) may be detected by incorporating the refined expression C' into the TBox as $C' \sqsubseteq \top$, and re-classifying to infer that, in fact, $\mathcal{T} \models C' \sqsubseteq \bot$. However, this is not a reasonable solution in the setting of concept induction for learning, as re-classification of a TBox containing many complex axioms will take too long considering it must be performed repeatedly for each of the many refinements produced in a generate-and-test based learning algorithm. Unless re-classification was possible in milliseconds or less, this approach would be practically infeasible. Although *incremental reasoning* algorithms are designed for fast re-classification when introducing new axioms, no such implementation currently exists which can perform as efficiently as we require [20]. Furthermore, no such algorithm exists for incremental classification relative to the closed-world interpretation $(\mathcal{I},\mathcal{U})$ that we aim to employ for learning complex concepts as discussed in Section 3.5.1 of Chapter 3.

Our approach to addressing this limitation is to analyse the closed-world interpretation $(\mathcal{I},\mathcal{U})$, which must be pre-computed prior to applying a refinement operator such as ρ, relative to the concepts in the TBox which are used in refinement. We aim to pre-compute a *static* set of axiomatic knowledge about the unsatisfiability of concept subexpressions being refined, such that it will be possible to detect, at the time of refinement, if a step may be unsatisfiable without resorting to further reasoning or coverage checking. We will show how to redefine the behaviour of an operator like ρ to incorporate such knowledge so that the refinements it produces are still generated with rewrite rules which are largely syntactic, but are restricted based on axiomatic knowledge to disallow certain refinements. The result is an operator that can quickly produce refinements that are used to traverse the space of concepts more efficiently than ρ, and that is more suited for application to learning over large knowledge bases.

4.2 Structuring the Interpretation for Learning

A closed-world interpretation such as $(\mathcal{I},\mathcal{U})$ describes how to interpret each of the components of a DL language like $\mathcal{SROIQ}$ as subsets of individuals from $\Delta^{(\mathcal{I},\mathcal{U})}$ which is fixed and finite. For example, a named concept $A \in N_C$ is mapped to a set $A^{(\mathcal{I},\mathcal{U})} \subseteq \Delta^{(\mathcal{I},\mathcal{U})}$. This can be viewed as a *global* interpretation of the concept A, in that it describes any and all individuals which are instances of A within the entire knowledge base. This interpretation may avail knowledge such as A is disjoint from concept B where $A^{(\mathcal{I},\mathcal{U})} \cap B^{(\mathcal{I},\mathcal{U})} = \emptyset$. A refinement operator such as ρ can leverage such information so that it never produces the refinement chain $A \rightsquigarrow A \sqcap B$ when $A \sqcap B$ is known to be unsatisfiable. Similarly, this refinement chain may be avoided if it is known that $A \sqsubseteq B$ where $A^{(\mathcal{I},\mathcal{U})} \subseteq B^{(\mathcal{I},\mathcal{U})}$, as such a refinement would be improper since $A \sqcap B \equiv A$.

Consider the case where the interpretation of concepts A and B overlap, where $A^{(\mathcal{I},\mathcal{U})} \cap B^{(\mathcal{I},\mathcal{U})} \neq \emptyset$ as illustrated in Figure 4.1.

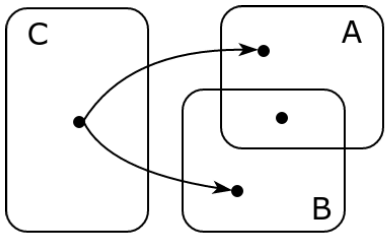


Figure 4.1: An example set of concepts A, B, C with two role tuples and where $A \sqcap B$ is satisfiable.

The concept $A \sqcap B$ is satisfiable because $(A \sqcap B)^{(\mathcal{I},\mathcal{U})}$ is non-empty, so a refinement operator may permit the refinement step $A \rightsquigarrow A \sqcap B$. However, consider this refinement in the context of the filler concept of a role expression such as $\exists r.A \rightsquigarrow \exists r.(A \sqcap B)$. In this case, as there is no r-successor which is an instance of $A \sqcap B$, the expression $\exists r.(A \sqcap B)$ is unsatisfiable with respect to $(\mathcal{I},\mathcal{U})$. However, the refinement step would have been permitted by ρ because $A \sqcap B$ is satisfiable with respect to $(\mathcal{I},\mathcal{U})$.

For any knowledge base containing many concept and role names, the space of all possible concepts which can be composed with an expressive DL such as $\mathcal{SROIQ}(D)$ can be vast. Learning algorithms which employ refinement operators to search concept spaces are typically very computationally expensive, as not only are many concepts generated, but computing their coverage for testing can also be expensive if the knowledge base contains a large amount of data. We are therefore motivated to reduce the number of concept expressions generated by a refinement operator which are unsuitable, such as concepts which are unsatisfiable, or refinement steps which are improper and produce equivalent concepts which unnecessarily inflate the search space.

To achieve this, we aim to utilise the distribution of known data amongst certain concepts in the knowledge base computable as the closed-world interpretation $(\mathcal{I},\mathcal{U})$ to guide refinement away from unsuitable candidates. Primarily, we aim to determine when certain refinement steps would otherwise lead to an unsatisfiable or equivalent expression, so as to prune these from the overall search space of concepts. As refinement operates over any part of a candidate expression, we need to formally define these as the *subexpressions* of concepts as follows.

Definition 4.2.1. ***(Concept Subexpressions)*** *All subexpressions of a concept expression C are defined inductively with function $sub(C)$ which maps concepts to sets where:*

$$sub(C) = \{C\} \cup \begin{cases} \{D \mid \forall C_{1 \le i \le n} : D \in sub(C_i)\} & \text{if } C = C_1 \sqcap \ldots \sqcap C_n \\ \{D \mid \forall C_{1 \le i \le n} : D \in sub(C_i)\} & \text{if } C = C_1 \sqcup \ldots \sqcup C_n \\ \{E \mid E \in sub(D)\} & \text{if } C = \Diamond r.D \end{cases}$$

for any role quantifier $\Diamond$. A ***subexpression*** *C' of concept C is any $C' \in sub(C)$.*

Example 4.2.2. *Consider the following concept expression S along with various numbered subexpressions:*

$$S = \overbrace{\exists r.(\underbrace{A \sqcap B}_{1})}^{3} \sqcap \overbrace{{}^{\leqslant 3}p.(\underbrace{A \sqcap B}_{2})}^{4}$$

In the derivation of $S^{(\mathcal{I},\mathcal{U})}$, the interpretation $(A \sqcap B)^{(\mathcal{I},\mathcal{U})}$ is used twice, once each in the contexts of subexpression (1) and (2). If $(A \sqcap B)^{(\mathcal{I},\mathcal{U})} \neq \emptyset$, then a refinement operator such as ρ_B must have produced this expression because subexpressions (1) and (2) may also be satisfiable. However, consider the case similar to that shown in Figure 4.1 where no r-successor is an instance of $A \sqcap B$. In this case, subexpression (3) is unsatisfiable, as is S. Alternatively, consider the case where there are no individuals with at most three p-successors *in $A \sqcap B$, which would make subexpression (4) unsatisfiable, and S also. In both of these cases, refining to the subexpression $A \sqcap B$ of (1), (2) would have produced an unsatisfiable expression overall, however without knowing that either role expression would be unsatisfiable with the filler concept $A \sqcap B$, a refinement operator such as ρ_B would generate subexpressions (1) and (2) on the basis that it is satisfiable under $(\mathcal{I}, \mathcal{U})$.*

For any knowledge base, we aim to derive information about the *distribution* of individuals, literals and role tuples of $(\mathcal{I}, \mathcal{U})$ to identify when, for example, refinement to the expression $A \sqcap B$ should be avoided because it is either unsatisfiable or improper as some subexpression of another concept expression. This information will then be used in the definition of a new refinement operator similar to ρ_B to control whether or not it chooses particular refinements to explicitly avoid generating concepts which do not progress a search towards solutions and otherwise waste valuable computational resources.

In order to achieve this, we aim to develop a method of pre-computing the set of individuals or literals which reside in the closed-world interpretation of *subexpressions* of concepts encountered in a learning as search problem. This will provide us with the means by which to identify expressions which can be safely pruned from a search. We begin with a method for identifying particular subexpressions of a DL concept expression, delineated by subexpressions which are the fillers of role expressions, which we refer to as *role subexpression contexts*.

Definition 4.2.3. ***(Role Subexpression Context)*** *A* ***role subexpression context****, also now referred to simply as a* ***context****, identifies a subexpression C_n of a tree-structured DL concept C by decomposition into a finite list of concept fragments λ of length $n \geq 1$ concept expressions $[C_1, \ldots, C_n]$ where each C_i for $1 \leq i \leq n-1$ takes the form of:*

- *A quantified role expression $\Diamond r.(\circ_i)$; or*
- *A quantified role expression in a conjunction $D_1 \sqcap \ldots \sqcap D_k \sqcap \Diamond r.(\circ_i)$; or*
- *A quantified role expression in a disjunction $D_1 \sqcup \ldots \sqcup D_k \sqcup \Diamond r.(\circ_i)$*

for $k \geq 1$, any quantifier $\Diamond$, role name $r \in N_R$ and subexpression symbol $\circ_i$. To reconstruct C from λ, we replace each subexpression symbol $\circ_i$ in C_i with C_{i+1} from $i = n-1$ to $i = 1$, collapsing the list from right to left to produce C.

Example 4.2.4. *Consider the following DL expression over concept names A, B, C, D and* *roles names r, s, d:*

$$\overbrace{D \sqcap {}^{\geqslant 2}r.(\overbrace{\exists d.(\underbrace{double[\geq 5.1]}_{\lambda_3}) \sqcup B}^{\lambda_2}) \sqcap \forall s.(\underbrace{A \sqcap C}_{\lambda_4})}^{\lambda_1}$$

All role subexpression contexts for this expression are those labelled λ_i for $1 \leq i \leq 4$ where:

$$\begin{aligned}
\lambda_1 &= [D \sqcap {}^{\geqslant 2}r.(\exists d.(double[\geq 5.1]) \sqcup B) \sqcap \forall s.(A \sqcap C)] \\
\lambda_2 &= [D \sqcap {}^{\geqslant 2}r.(\circ_1) \sqcap \forall s.(A \sqcap C), \exists d.(double[\geq 5.1]) \sqcup B] \\
\lambda_3 &= [D \sqcap {}^{\geqslant 2}r.(\circ_1) \sqcap \forall s.(A \sqcap C), \exists d.(\circ_2) \sqcup B, double[\geq 5.1]] \\
\lambda_4 &= [D \sqcap {}^{\geqslant 2}r.(\exists d.(double[\geq 5.1]) \sqcup B) \sqcap \forall s.(\circ_1), A \sqcap C]
\end{aligned}$$

Example 4.2.4 illustrates how various subexpressions nested within roles of a tree-structured $\mathcal{SROIQ}(D)$ concept expression can be referred to with a role subexpression context λ. To reconstruct a concept expression C from a role subexpression context λ, we define functions to replace a subexpression identified with symbol $\circ$ with some other DL expression, and a nesting function which can collapse a list λ to the single expression C.

$$repl_\circ(C, D) = \begin{cases} \Diamond r.D & \text{if } C = \Diamond r.(\circ) \\ C_1 \sqcap \ldots \sqcap C_n \sqcap \Diamond r.D & \text{if } C = C_1 \sqcap \ldots \sqcap C_n \sqcap \Diamond r.(\circ)\ (n \geq 1) \\ C_1 \sqcup \ldots \sqcup C_n \sqcup \Diamond r.D & \text{if } C = C_1 \sqcup \ldots \sqcup C_n \sqcup \Diamond r.(\circ)\ (n \geq 1) \end{cases}$$

$$nest(\lambda) = \begin{cases} C & \text{if } \lambda = [C] \\ nest([C_1, \ldots, C'_{n-1}]) \text{ where} & \text{if } \lambda = [C_1 \ldots, C_n]\ (n \geq 2) \\ \qquad\qquad C'_{n-1} = repl_\circ(C_{n-1}, C_n) & \end{cases}$$

We also define the set of individuals or literals that may occur as instances of the innermost subexpression C_n referred to by $\lambda = [C_1, \ldots, C_n]$ for $n \geq 1$ as a *local domain*.

Definition 4.2.5. ***(Local Domain)*** *A* ***local domain*** *Δ_λ is the set of all individuals (literals) which may occur in the closed-world interpretation $(\mathcal{I}, \mathcal{U})$ of the subexpression C_n denoted by the role subexpression context $\lambda = [C_1, \ldots, C_n]$ for $n \geq 1$ as $\Delta_\lambda = S_n$ where S_k for*

$1 \leq k \leq n$ is defined inductively as follows:

$$S_k = \begin{cases} C_1^{(\mathcal{I},\mathcal{U})} & \text{if } k = 1 \\ \{j \mid \forall \langle i,j \rangle \in r^{(\mathcal{I},\mathcal{U})} : i \in (C'_{k-1})^{(\mathcal{I},\mathcal{U})} \cap S_{k-1} \wedge j \in (C'_k)^{(\mathcal{I},\mathcal{U})}\} & \text{if } 2 \leq k \leq n \\ \quad \text{where } C'_k = nest([C_k, \dots, C_n]), \text{ and where} & \\ \quad C'_{k-1} = \begin{cases} \Diamond r.(C'_k) & \text{if } C_{k-1} = \Diamond r.(\circ_{k-1}) \\ \Diamond r.(C'_k) & \text{if } C_{k-1} = D_1 \sqcup \dots \sqcup D_m \sqcup \Diamond r.(\circ_{k-1}) \\ D_1 \sqcap \dots \sqcap D_m \sqcap \Diamond r.(C'_k) & \text{if } C_{k-1} = D_1 \sqcap \dots \sqcap D_m \sqcap \Diamond r.(\circ_{k-1}) \end{cases} & \end{cases}$$

where $m \geq 1$.

Example 4.2.6. *Consider again the role subexpression contexts labelled λ_i for $1 \leq i \leq 4$ of the following concept expression from Example 4.2.4:*

$$\overbrace{D \sqcap {}^{\geqslant 2}r.(\overbrace{\exists d.(\underbrace{double[\geq 5.1]}_{\lambda_3}) \sqcup B}^{\lambda_2}) \sqcap \forall s.(\underbrace{A \sqcap C}_{\lambda_4})}^{\lambda_1}$$

We then have the following descriptions of the local domains:

- *Δ_{λ_1}: All instances in the set $(D \sqcap {}^{\geqslant 2}r.(\exists d.(double[\geq 5.1]) \sqcup B) \sqcap \forall s.(A \sqcap C))^{(\mathcal{I},\mathcal{U})}$.*
- *Δ_{λ_2}: All r-successors which are instances of $\exists d.(double[\geq 5.1]) \sqcup B$ where there are at least two per instance of $D \sqcap \forall s.(A \sqcap C)$.*
- *Δ_{λ_3}: All d-successors which are double valued literals greater than or equal to 5.1 where there is at least one per instance of Δ_{λ_2}.*
- *Δ_{λ_4}: All s-successors of instances of $D \sqcap {}^{\geqslant 2}r.(\exists d.(double[\geq 5.1]) \sqcup B)$ where every s-successor is an instance of $A \sqcap C$.*

Intuitively, each local domain Δ_λ represents the set of individuals or literals which can lie in the closed-world interpretation of the subexpression C_n referred to by $\lambda = [C_1, \dots, C_n]$, and is always a subset of the *global* closed-world domain where $\Delta_\lambda \subseteq \Delta^{(\mathcal{I},\mathcal{U})}$. Relative to each *local* domain Δ_λ in context λ, the subsumption relationships between concept expressions may be different.

Example 4.2.7. *Consider the concepts A, B where $B^{(\mathcal{I},\mathcal{U})} \subseteq A^{(\mathcal{I},\mathcal{U})}$, or when they overlap such that $A^{(\mathcal{I},\mathcal{U})} \cap B^{(\mathcal{I},\mathcal{U})} \neq \emptyset$, with both cases illustrated in Figure 4.2.*

Consider the contexts λ_i for $1 \leq i \leq 9$ and their associated local domains Δ_{λ_i} each of which contain one or more instances of $A^{(\mathcal{I},\mathcal{U})} \cup B^{(\mathcal{I},\mathcal{U})}$. The sets which represent the

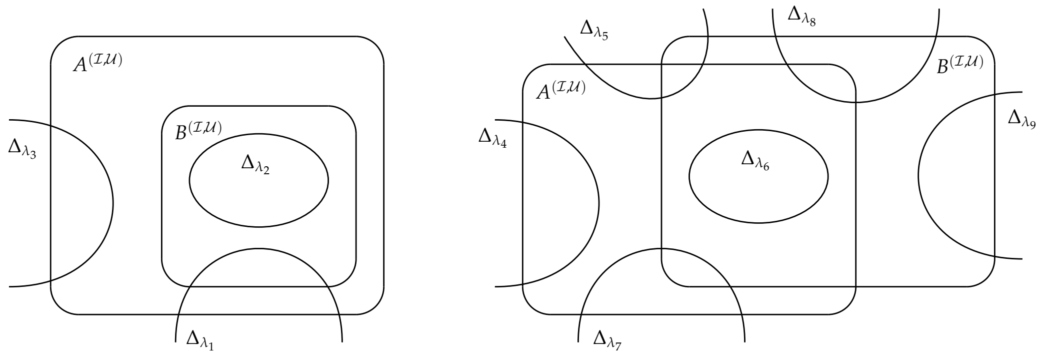


Figure 4.2: Two subsumption relationships between concepts A, B, the left where $B^{(\mathcal{I},\mathcal{U})} \subseteq A^{(\mathcal{I},\mathcal{U})}$ and the right where $A^{(\mathcal{I},\mathcal{U})}$ and $B^{(\mathcal{I},\mathcal{U})}$ overlap. This represents the two cases where the concept $A \sqcap B$ is satisfiable as $A^{(\mathcal{I},\mathcal{U})} \cap B^{(\mathcal{I},\mathcal{U})} \neq \emptyset$. For each of these two cases, intersection with various possible *local domains* Δ_{λ_i} for $1 \leq i \leq 9$ are shown.

intersection $\Delta_{\lambda_i} \cap (A \sqcup B)^{(\mathcal{I},\mathcal{U})}$ are illustrated as the nine possibilities numbered in Figure 4.2, where we denote the intersection of either $A^{(\mathcal{I},\mathcal{U})}$ or $B^{(\mathcal{I},\mathcal{U})}$ with the local domain as $A_\lambda^{(\mathcal{I},\mathcal{U})} = A^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$ and $B_\lambda^{(\mathcal{I},\mathcal{U})} = B^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$. Depending on the subsumption relationship between $A^{(\mathcal{I},\mathcal{U})}$ and $B^{(\mathcal{I},\mathcal{U})}$, we summarise whether the concept $A \sqcap B$ is equivalent to A or B relative to each local domain Δ_{λ_i} as follows.

***i* for λ_i and Δ_{λ_i}**	***Relationship***	***Equivalences in context λ_i relative to Δ_{λ_i}***
1, 7	$B_\lambda^{(\mathcal{I},\mathcal{U})} \subset A_\lambda^{(\mathcal{I},\mathcal{U})}$	$A \sqcap B \equiv B$
2, 6	$B_\lambda^{(\mathcal{I},\mathcal{U})} = A_\lambda^{(\mathcal{I},\mathcal{U})}$	$A \sqcap B \equiv A$ *and* $A \sqcap B \equiv B$
8	$B_\lambda^{(\mathcal{I},\mathcal{U})} \supset A_\lambda^{(\mathcal{I},\mathcal{U})}$	$A \sqcap B \equiv A$
3, 4, 9	$A_\lambda^{(\mathcal{I},\mathcal{U})} \cap B_\lambda^{(\mathcal{I},\mathcal{U})} = \emptyset$	$A \sqcap B \equiv \bot$
5	$A_\lambda^{(\mathcal{I},\mathcal{U})} \cap B_\lambda^{(\mathcal{I},\mathcal{U})} \neq \emptyset$	$A \sqcap B \not\equiv A$ *and* $A \sqcap B \not\equiv B$

The set of instances covered by the subexpression $A \sqcap B$ in the context of each λ_i may be different depending on the individuals in the local domain Δ_{λ_i}, and only in the case of Δ_{λ_5} is the expression $A \sqcap B$ not equivalent to one of: A, B, or $\bot$. Therefore, we would expect that in applying a refinement operator to specialise either subexpression A or B in context λ_5, that $A \sqcap B$ should be permitted because it is not equivalent to A, B or $\bot$, whereas in all other cases, at least one of these equivalences occur. Where $A \equiv A \sqcap B$ or $B \equiv A \sqcap B$ holds, such a specialisation would be improper, and where $A \sqcap B \equiv \bot$ holds, such a specialisation may result in an unsatisfiable expression.

The interpretation of subsumption relationships as shown in Example 4.2.7 between two concepts A, B which share instances with a local domain Δ_λ gives rise to the notion of *context-specific interpretations* relative to the closed-world interpretation $(\mathcal{I},\mathcal{U})$ for some context λ, as we will now describe in the next section.

4.2.1 Context-Specific Interpretations

A *context-specific interpretation* is an interpretation of some concept expression relative to a context λ and local domain Δ_λ. The motivation behind defining such an interpretation is to provide a means to reveal the subsumption relationships between concepts relative to a local domain, which as we have demonstrated in the last section may be different to the relationships implied under the closed-world interpretation $(\mathcal{I},\mathcal{U})$. We intend to utilise such concept subsumption information in the definition of a refinement operator which modifies concept subexpressions, so that it can recognise when to avoid generating certain refinement steps which result in producing concepts which are not useful to the search.

Definition 4.2.8. ***(Context-Specific Interpretation)*** *Given a closed-world interpretation $(\mathcal{I},\mathcal{U})$, a* ***context-specific interpretation*** *$\mathcal{J}_\lambda$ is defined as the tuple $\mathcal{J}_\lambda = (\cdot^{\mathcal{J}_\lambda}, \Delta_{\lambda_1}, \ldots, \Delta_{\lambda_k})$ for subexpression contexts λ_j for $1 \leq j \leq k$ where each Δ_{λ_j} is a local domain for context λ_j, and where $\cdot^{\mathcal{J}_\lambda}$ is a function which maps atomic concepts A relative to any context λ to subsets of Δ_λ, and roles r relative to any context λ to subsets of $\Delta_\lambda \times \Delta^{(\mathcal{I},\mathcal{U})}$ as follows:*

$$A^{\mathcal{J}_\lambda} = A^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$$
$$r^{\mathcal{J}_\lambda} = \{\langle i,j\rangle \mid \langle i,j\rangle \in r^{(\mathcal{I},\mathcal{U})} \text{ s.t. } i \in \Delta_\lambda\}$$

The interpretation of complex concepts by $\mathcal{J}_\lambda$ relative to any context λ is defined similarly to the IC-interpretation as per Definition 3.5.2 as follows, where

$$\begin{array}{rcl}
\top^{\mathcal{J}_\lambda} & = & \Delta_\lambda \\
\bot^{\mathcal{J}_\lambda} & = & \emptyset \\
(\neg C)^{\mathcal{J}_\lambda} & = & \Delta_\lambda \setminus C^{\mathcal{J}_\lambda} \\
(C \sqcap D)^{\mathcal{J}_\lambda} & = & C^{\mathcal{J}_\lambda} \cap D^{\mathcal{J}_\lambda} \\
(C \sqcup D)^{\mathcal{J}_\lambda} & = & C^{\mathcal{J}_\lambda} \cup D^{\mathcal{J}_\lambda} \\
\{i\}^{\mathcal{J}_\lambda} & = & \{i \mid i \in \Delta_\lambda\} \\
(\exists r.C)^{\mathcal{J}_\lambda} & = & \{i \mid i \in \Delta_\lambda \text{ s.t. } \exists j.\langle i,j\rangle \in r^{\mathcal{J}_\lambda} \wedge j \in C^{(\mathcal{I},\mathcal{U})}\} \\
(\forall r.C)^{\mathcal{J}_\lambda} & = & \{i \mid i \in \Delta_\lambda \text{ s.t. } \forall j.\langle i,j\rangle \in r^{\mathcal{J}_\lambda} \to j \in C^{(\mathcal{I},\mathcal{U})}\} \\
({\geqslant^n} r.C)^{\mathcal{J}_\lambda} & = & \{i \mid i \in \Delta_\lambda \text{ s.t. } \sharp\{j.\langle i,j\rangle \in r^{\mathcal{J}_\lambda} \wedge j \in C^{(\mathcal{I},\mathcal{U})}\} \geqslant n\} \\
({\leqslant^n} r.C)^{\mathcal{J}_\lambda} & = & \{i \mid i \in \Delta_\lambda \text{ s.t. } \sharp\{j.\langle i,j\rangle \in r^{\mathcal{J}_\lambda} \wedge j \in C^{(\mathcal{I},\mathcal{U})}\} \leqslant n\}
\end{array}$$

This context-specific interpretation is defined in such a way that, for any concept C, it will be the case that $C^{\mathcal{J}_\lambda} \subseteq C^{(\mathcal{I},\mathcal{U})}$ as $C^{\mathcal{J}_\lambda}$ is the $(\mathcal{I},\mathcal{U})$ interpretation of C relative to a local domain Δ_λ in subexpression context λ, and where each $\Delta_\lambda \subseteq \Delta^{(\mathcal{I},\mathcal{U})}$ for any context λ.

A context-specific interpretation $\mathcal{J}_\lambda$ relative to some context λ has some interesting properties which make it useful for inducing concepts with refinement operators. Most importantly, the set of inclusions $C \sqsubseteq D$ relative to a local domain Δ_λ modelled by $\mathcal{J}_\lambda$ denoted $C \sqsubseteq_{\mathcal{J}_\lambda} D$ may be tighter than those axioms implied by a TBox $\mathcal{T}$ for the whole knowledge-base $\mathcal{K}$ as illustrated in Example 4.2.7. We therefore associate each context λ with a number of inclusions which we denote *local axioms*, as follows.

Definition 4.2.9. ***(Local Axioms)*** *Given two concepts C, D relative to a subexpression context λ and a closed-world interpretation $\mathcal{J}_\lambda$,* ***local axioms*** *denoted $\mathcal{T}_\lambda$ are the set of all axioms of the form:*

- *Equivalence: $C \equiv_{\mathcal{J}_\lambda} D$ where $C^{\mathcal{J}_\lambda} = D^{\mathcal{J}_\lambda}$*
- *Strict subsumption: $C \sqsubset_{\mathcal{J}_\lambda} D$ where $C^{\mathcal{J}_\lambda} \subset D^{\mathcal{J}_\lambda}$*
- *Disjointness: $C \sqcap D \sqsubseteq_{\mathcal{J}_\lambda} \perp$ where $C^{\mathcal{J}_\lambda} \cap D^{\mathcal{J}_\lambda} = \emptyset$*

Example 4.2.10. *Consider concepts A, B in a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $B \sqsubseteq A \in \mathcal{T}$, and where $\mathcal{A} \neq \emptyset$ and $\mathcal{K}$ is consistent. Therefore, all interpretations $\mathcal{I}$ which are models of $\mathcal{K}$ have $B^{\mathcal{I}} \subseteq A^{\mathcal{I}}$. Consider the closed-world interpretation $(\mathcal{I},\mathcal{U})$ which models $B^{(\mathcal{I},\mathcal{U})} \subseteq A^{(\mathcal{I},\mathcal{U})}$ over $\mathcal{K}$, and a context-specific interpretation $\mathcal{J}_\lambda$ relative to some local domain Δ_λ. Under $\mathcal{J}_\lambda$, it is possible that any of the following may hold:*

- *$\mathcal{T}_\lambda \models B \equiv A$ where $A^{\mathcal{J}_\lambda} = B^{\mathcal{J}_\lambda}$*
- *$\mathcal{T}_\lambda \models B \sqcap A \sqsubseteq \perp$ where $B^{\mathcal{J}_\lambda} = \emptyset$ (B is unsatisfiable in local domain Δ_λ)*
- *$\mathcal{T}_\lambda \models B \sqsubset A$ where $B^{\mathcal{J}_\lambda} \cap A^{\mathcal{J}_\lambda} = B^{\mathcal{J}_\lambda}$*

Note while each of these interpretations are consistent with $B^{(\mathcal{I},\mathcal{U})} \subseteq A^{(\mathcal{I},\mathcal{U})}$, the first case recognises that, in the context of λ, a refinement chain $A \rightsquigarrow B$ is improper. Similarly, the second case where $B \equiv_{\mathcal{J}_\lambda} \perp$ can be used to avoid expressions containing B in the context of λ if it will result in the production of an unsatisfiable concept.

Proposition 4.2.11. *For all inclusion axioms ϕ in the set of local axioms $\mathcal{T}_\lambda$ by Definition 4.2.9 for some context λ, if $\mathcal{T} \models \phi$ then $\mathcal{T}_\lambda \models \phi$ for the TBox $\mathcal{T}$ which models inclusion axioms ϕ relative to the closed-world interpretation $(\mathcal{I},\mathcal{U})$.*

Proof. We prove Proposition 4.2.11 for each form of local axiom $\phi \in \mathcal{T}_\lambda$ over any two concepts C, D which can be interpreted by $(\mathcal{I},\mathcal{U})$ and $\mathcal{J}_\lambda$ by noting that each context-specific interpretation $C^{\mathcal{J}_\lambda}$ and $D^{\mathcal{J}_\lambda}$ are subsets of $C^{(\mathcal{I},\mathcal{U})}$ and $D^{(\mathcal{I},\mathcal{U})}$ respectively as $C^{\mathcal{J}_\lambda} = C^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$ and $D^{\mathcal{J}_\lambda} = D^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$.

- For $\phi = (C \sqsubset D)$, we must show that the implication $C^{(\mathcal{I},\mathcal{U})} \subset D^{(\mathcal{I},\mathcal{U})} \rightarrow C^{\mathcal{J}_\lambda} \subset D^{\mathcal{J}_\lambda}$ always holds for any subexpression context λ. The implication fails only if the antecedent $C^{(\mathcal{I},\mathcal{U})} \subset D^{(\mathcal{I},\mathcal{U})}$ holds and the consequent $C^{\mathcal{J}_\lambda} \subset D^{\mathcal{J}_\lambda}$ does not. As $C^{\mathcal{J}_\lambda}$ is always a subset of $C^{(\mathcal{I},\mathcal{U})}$ where $C^{\mathcal{J}_\lambda} = C^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$, it must be the case that $C^{\mathcal{J}_\lambda} \subset D^{(\mathcal{I},\mathcal{U})}$. In the case where $C^{\mathcal{J}_\lambda} = \emptyset$, the consequent holds trivially. Now consider there exists an individual $i \in C^{\mathcal{J}_\lambda}$, and therefore we also know that $i \in D^{(\mathcal{I},\mathcal{U})}$. The consequent fails when it can be shown that $i \notin D^{\mathcal{J}_\lambda}$. If we assume $i \notin D^{\mathcal{J}_\lambda}$, then it must be the case that $i \notin D^{(\mathcal{I},\mathcal{U})}$, which is a contradiction. Therefore, it must be the case that $i \in D^{\mathcal{J}_\lambda}$ which means $C^{\mathcal{J}_\lambda} \subset D^{\mathcal{J}_\lambda}$ also holds for any individual i.
- For $\phi = (C \equiv D)$, we must show that $C^{(\mathcal{I},\mathcal{U})} = D^{(\mathcal{I},\mathcal{U})} \rightarrow C^{\mathcal{J}_\lambda} = D^{\mathcal{J}_\lambda}$ for any subexpression context λ. This holds trivially as $C^{\mathcal{J}_\lambda}$ and $D^{\mathcal{J}_\lambda}$ are the same subset in the intersection $C^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$ and $D^{(\mathcal{I},\mathcal{U})} \cap \Delta_\lambda$.
- For $\phi = (C \sqcap D \sqsubseteq \bot)$, we must show that $C^{(\mathcal{I},\mathcal{U})} \cap D^{(\mathcal{I},\mathcal{U})} = \emptyset \rightarrow C^{\mathcal{J}_\lambda} \cap D^{\mathcal{J}_\lambda} = \emptyset$. This also holds trivially as there are no common subsets of $C^{(\mathcal{I},\mathcal{U})}$ and $D^{(\mathcal{I},\mathcal{U})}$ which are not disjoint.

Therefore, we conclude that for any local axiom ϕ in $\mathcal{T}_\lambda$ it is the case that if $\mathcal{T} \models \phi$ then $\mathcal{T}_\lambda \models \phi$ for any context λ. $\square$

Example 4.2.10 and Proposition 4.2.11 demonstrate that any inclusion axioms $C \sqsubseteq D \in \mathcal{T}$ where $C^{(\mathcal{I},\mathcal{U})} \subseteq D^{(\mathcal{I},\mathcal{U})}$ are not contradicted under $\mathcal{J}_\lambda$ relative to a context λ as it is always the case that $C^{\mathcal{J}_\lambda} \subseteq D^{\mathcal{J}_\lambda}$, even if it can be shown that under $\mathcal{J}_\lambda$ that $C \equiv_{\mathcal{J}_\lambda} D$ or $C \cap D \sqsubseteq_{\mathcal{J}_\lambda} \bot$ hold, yet this cannot be shown under $(\mathcal{I},\mathcal{U})$. While relationships such as $C \sqsubseteq D$ are typically inferred with logical reasoning algorithms under the open-world assumption relative to the standard first-order interpretation $\mathcal{I}$, they can be computed explicitly over a fixed model such as $(\mathcal{I},\mathcal{U})$ under the closed-world assumption for any two concepts by testing the relationship between sets $C^{(\mathcal{I},\mathcal{U})}$ and $D^{(\mathcal{I},\mathcal{U})}$ composed of asserted data, and similarly, under $\mathcal{J}_\lambda$ between $C^{\mathcal{J}_\lambda}$ and $D^{\mathcal{J}_\lambda}$ for any subexpression context λ.

Unfortunately, for any knowledge base consisting of many concept terms, role terms and individuals, it would be practically infeasible to enumerate all possible concepts expressible in a concept language like $\mathcal{SROIQ}(D)$, along with their subexpression contexts λ, in order to pre-compute $\mathcal{J}_\lambda$ in its entirety, unlike $(\mathcal{I},\mathcal{U})$. Furthermore, even if it is possible to enumerate each concept and subexpression context, the computation of each local domain Δ_λ and the context-specific interpretation of all concept terms is again likely to be infeasible especially given a large data set of individuals and literals.

Recall that our aim is to define and construct a context-specific interpretation to provide a refinement operator with information about which concepts to prune when performing refinements of concept subexpressions which do not aid in the search. With this aim in mind, we intend to describe how to construct a particular finite fixed context-specific interpretation $\mathcal{J}_\lambda$ based on $(\mathcal{I},\mathcal{U})$ which is small enough to compute reasonably quickly, but which may still be used to permit a refinement operator to take advantage of knowledge which $\mathcal{J}_\lambda$ affords relative to a limited set of contexts. Specifically, the contexts we will describe consist exclusively of single role expressions in conjunction with *simple concepts* including $\top$, atomic concept names and negated atomic concept names. At the very least, such contexts will subsume more complex subexpressions composed of such fragments, and can be used to identify certain cases where refinements would lead to concepts which can clearly be avoided. As we will describe in the next section, such a limited interpretation still affords new knowledge which can be effectively leveraged by a refinement operator to reduce the search space of concepts. We will begin by describing a method for constructing such a limited context-specific interpretation, and then describe how it can be incorporated directly into the definition of a new pair of downward and upward refinement operators $\rho_{\bar{\lambda}}$ and $\upsilon_{\bar{\lambda}}$ which are defined in terms of a set of applicable contexts $\bar{\lambda}$.

4.2.2 The Context Graph

A *context graph* is a data structure which roughly captures a context-specific interpretation $\mathcal{J}_\lambda$ for some knowledge base $\mathcal{K}$ based on its closed-world interpretation $(\mathcal{I},\mathcal{U})$. The context graph represents a restricted and finite collection of contexts λ and associates with each a set of information about certain concepts which are satisfiable in the context. In doing so, the context graph reveals local axiomatic information about the distribution of $(\mathcal{I},\mathcal{U})$ relative to a finite number of particular concept expressions and their subexpressions.

Definition 4.2.12. ***(Context Graph)*** *A **context graph** $\mathfrak{G}$ is a graph structure $\mathfrak{G} = (V, E)$ resembling a set of trees where:*

- *V is the set of vertices $V = \{\lambda_1, \ldots, \lambda_m\}$ where each vertex label λ_i for $1 \leq i \leq m$ is a role subexpression context. Each vertex V with label λ is associated with a set Δ_{λ_i} which captures all individuals (literals) which comprise a local domain for λ_i.*

- *E is the set of directed edges $E = \{(\lambda_1, \lambda'_1), \ldots, (\lambda_n, \lambda'_n)\}$ where each edge label (λ_i, λ'_i) for $1 \leq i \leq n$ represents the edge from vertex with label λ_i to λ'_i where $\lambda_i = [C_1, \ldots, C_k]$ for $k \geq 1$ and $\lambda'_i = [C_1, \ldots, C_k \sqcap \Diamond r.(\circ_k), C_{k+1}]$ for some $r \in N_R$, concept expression C_{k+1} and role quantifier $\Diamond$. Each edge (λ_i, λ'_i) is associated with the set $(\Diamond r.(C_{k+1}))^{\mathcal{I}_{\lambda_i}}$ to capture the context-specific interpretation of the role expression in context λ_i.*

A context graph $\mathfrak{G}$ is defined over contexts $\lambda = [C_1, \ldots, C_n]$ where any C_i for $1 \leq i \leq n$ is composed exclusively of atomic and negated atomic concept symbols $\alpha = \{\top, \bot\} \cup \{A, \neg A, \mid A \in N_C\}$ or role expressions as follows:

$$C_i \in \begin{cases} \{A \sqcap \Diamond r.(\circ_i) \mid A \in \alpha, r \in N_R, \Diamond \in \{\exists, \forall, {\geqslant}n, {\leqslant}n\}\} & \text{if } 1 \leq i \leq n-1 \\ \alpha & \text{if } i = n \end{cases}$$

In this way, the context graph is restricted to capture a limited set of possible subexpression contexts λ. As discussed in Section 4.2.1, this restriction enables the graph to be computable in practice, as computing the graph for all possible concepts and their subexpressions is infeasible. However, as we will later show, the context graph can be used to provide useful information such as local axioms about a broader space of concepts that are available to a refinement operator.

$$\begin{tikzcd}
 & & {[\top]} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & {[\top \sqcap \exists r.(\circ_1), \top]} \arrow[dl] \arrow[d] & {[\top \sqcap \exists r.(\circ_1), B]} & {[\top \sqcap \exists t.(\circ_1), \neg C]} \arrow[d] \\
 {\begin{array}{c}[\top \sqcap \exists r.(\circ_1), \\ \top \sqcap \exists s.(\circ_2), \top]\end{array}} & {\begin{array}{c}[\top \sqcap \exists r.(\circ_1), \\ \top \sqcap {\geqslant}2 s.(\circ_2), A]\end{array}} & & {\begin{array}{c}[\top \sqcap \exists t.(\circ_1), \\ \neg C \sqcap \forall u.(\circ_2), \top]\end{array}}
\end{tikzcd}$$

Figure 4.3: An example *context graph* structure showing vertex labels only (see Definition 4.2.12). A context graph represents a collection of directed trees which capture the set of role subexpression contexts which are permissible for any context λ by conjunction with an atomic or atomic negated concept and quantified role expressions. The context graph is designed for use in concept refinement as it indicates to a refinement operator which concepts in some subexpression context λ are available for refinement as the set of immediate child nodes in the graph. While only a single tree structure is shown here, there may be several in the entire context graph rooted at different contexts for each named concept $A \in N_C$ in the form of $\lambda = [A]$ or $\lambda = [\neg A]$.

A context graph can be computed by way of a procedure known as an *instance chase*. Before we describe the algorithm for performing an instance chase, we define several terms which will aid in its explanation. The instance chase begins with a set of labelled examples $\mathcal{E}$ with labels Ω where $\mathcal{E} = \bigcup_{\forall \omega \in \Omega} \mathcal{E}^{\omega}$ for learning where $\mathcal{E} \subseteq N_I$, and constructs *instance chains* given the closed-world interpretation of every role $r \in N_R$.

Definition 4.2.13. ***(Instance Chain)*** *Given a knowledge base $\mathcal{K}$, a set of named individuals N_I, datatype literals $N_\mathcal{D}$, role names N_R and a closed-world interpretation $(\mathcal{I},\mathcal{U})$, an* ***instance chain*** $\mathfrak{ic}$ *is a sequence of n tuples of the form*

$$\mathfrak{ic} = [\langle a_0, a_1 \rangle, \langle a_1, a_2 \rangle, \ldots, \langle a_{n-2}, a_{n-1} \rangle, \langle a_{n-1}, a_n \rangle]$$

where $a_i, a_{i+1} \in N_I$ for $1 \leq i \leq n-1$, and where $a_n \in N_I$ or $a_n \in N_\mathcal{D}$, and each tuple $\langle a_i, a_j \rangle \in r^{(\mathcal{I},\mathcal{U})}$ for any role $r \in N_R$. An instance chain is ***acyclic*** *if, for any a_j of any tuple $\langle a_i, a_j \rangle$ in $\mathfrak{ic}$, a_j does not also appear in any tuple $\langle a_j, a_k \rangle$ in $\mathfrak{ic}$. We say that an individual or literal a_j is* ***reachable*** *from individual a_i if there exists any instance chain which starts with tuple $\langle a_i, a_{i+1} \rangle$ and ends with tuple $\langle a_{j-1}, a_j \rangle$). We say that any a_i is* ***contained in*** *an instance chain $\mathfrak{ic}$ if it appears in any tuple $\langle a_i, a_j \rangle$ or $\langle a_j, a_i \rangle$ in $\mathfrak{ic}$.*

The construction of a context graph $\mathfrak{G}$ is performed by expanding so-called r-successors for every role $r \in N_R$ starting from the examples $e \in \mathcal{E}$.

Definition 4.2.14. ***(Role r-Successors, Predecessor)*** *Given a role $r \in N_R$,* ***predecessor*** *individual $i \in N_I$ and the interpretation $(\mathcal{I},\mathcal{U})$, the set of r-**successors** of i is the set $succ(i, r) = \{j \mid \langle i, j \rangle \in r^{(\mathcal{I},\mathcal{U})}\}$.*

When expanding r-successors for every individual encountered within an instance chain, we analyse the set of concepts which describe each individual to construct subexpression contexts λ which form the vertices of the context graph. In this way, the context graph is comprised exclusively of concepts and roles which describe only those individuals and literals which are reachable via instance chains from the examples via role assertions in $\mathcal{A}$. As we aim to use the context graph to guide a refinement operator in the search for concept expressions, excluding irrelevant concept and role names effectively reduces the concept search space.

To understand how the context graph can aid in determining possible refinements, consider the concept expression $\exists r.(\top)$. There are several ways to downward-refine this concept, including:

$$\begin{array}{ll} \exists r.(\top) \rightsquigarrow \exists r.(\top) \sqcap C & \text{(conjunction with a concept)} \\ \exists r.(\top) \rightsquigarrow \exists r.(C) & \text{(refinement of the filler } \top \rightsquigarrow C\text{)} \\ \exists r.(\top) \rightsquigarrow {}^{\geqslant 2}r.(\top) & \text{(refinement of the quantifier)} \end{array}$$

In order to determine the set of concepts which can be used in conjunction with $\exists r.(\top)$, we can compute the set of concepts which describe predecessors of r-successors, or instances of $\exists r.(\top)$. For example, consider the individual i where $i \in (\exists r.(\top))^{(\mathcal{I},\mathcal{U})}$. The set of *simple concepts* describing i can be computed as $sc(\{i\})$ as follows:

$$\begin{array}{lll} sc(S) & = \{C \mid \forall i \in S, C \in atom(i)\} & \\ atom(i) & = atom_{+}(i) \cup atom_{\neg}(i) & \text{(all } \textit{simple concepts} \text{ describing } i\text{)} \\ atom_{+}(i) & = \{A \mid A \in N_C \text{ s.t. } i \in A^{(\mathcal{I},\mathcal{U})}\} & \text{(all concept names)} \\ atom_{\neg}(i) & = \{\neg A \mid A \in N_C \text{ s.t. } A \notin atom_{+}(i)\} & \text{(all negated concept names)} \end{array}$$

All concepts $C \in sc(\{i\})$ are those which may appear as $\exists r.(\top) \sqcap C$ such that C is satisfiable with respect to $(\mathcal{I},\mathcal{U})$. Similarly, we can generate the set of all simple concepts which the filler $\top$ may be downward refined to in $\exists r.(\top)$ by computing the set $sc(succ(i,r))$ for all r-successors of i, where for all $C \in sc(succ(i,r))$ we will find that $\exists r.(C)$ is also satisfiable under $(\mathcal{I},\mathcal{U})$, at least relative to the predecessor individual i. Note that this set may be different for other predecessor individuals. The context graph will reflect these differences by capturing satisfiable concept subexpressions as its vertices which were computed from individuals found along instance chains via r-successors.

Role quantifiers $\diamond$ for which concepts of the form $\diamond r.(D)$ are satisfiable for various simple concepts D can be computed by analysing the set of r-successors $S = succ(i,r)$ for some predecessor individual i, along with the set of concepts for which each successor $j \in S$ are instances as $sc(S)$. Pairs (D,n) which capture when there are exactly n r-successors which are instances of D from i can be computed by the role filler function $rf(S)$ as follows:

$$rf(S) \quad = \{(D,n) \mid \forall D \in sc(S), n = |S \sqcap D^{(\mathcal{I},\mathcal{U})}| \wedge n \geq 1\}$$

The pairs (D,n) in $rf(S)$ simply reflect that exactly n individuals in S are instances of D. If S was computed as the set of r-successors from i, then $rf(S)$ can be used to compute all satisfiable concepts of the form $\diamond r.(D)$ for various role quantifiers

$\Diamond \in \{\exists, \forall, {\geqslant}m, {\leqslant}m\}$ as follows:

- $\exists$: If $(D,n) \in rf(S)$, then $\exists r.(D)$ is satisfiable wrt. $(\mathcal{I},\mathcal{U})$;
- $\forall$: If $(D,n) \in rf(S)$ where $n = |S|$, then $\forall r.(D)$ is satisfiable wrt. $(\mathcal{I},\mathcal{U})$;
- ${\geqslant}m$: If $(D,n) \in rf(S)$, then ${\geqslant}m r.(D)$ for $1 \leq m \leq n$ is satisfiable wrt. $(\mathcal{I},\mathcal{U})$;
- ${\leqslant}m$: If $(D,n) \in rf(S)$, then ${\leqslant}m r.(D)$ for $m \geq n$ is satisfiable wrt. $(\mathcal{I},\mathcal{U})$.

Recall that the vertices of the context graph are the subexpression contexts λ reflecting various concept expressions and their subexpressions which are satisfiable under $(\mathcal{I},\mathcal{U})$. These expressions are restricted in form, such as $[C \sqcap \Diamond r.(\circ), D]$ which, when collapsed into an expression $nest(\lambda) = C \sqcap \Diamond r.(D)$, where C, D are simple concepts. In starting from the set of examples $\mathcal{E}$, we begin construction of the context graph by computing $sc(\mathcal{E})$ to give an initial set of vertices. For example, assume $sc(\mathcal{E}) = \{\top, C\}$. Then, the context graph will consist of two vertices, namely $\lambda_1 = [\top]$ and $\lambda_2 = [C]$. Then, from each example $e \in \mathcal{E}$, we compute all r-successors S for each role name $r \in N_R$ and the set of simple concepts $rf(S)$ for which they are instances. If say, example $e_0 \in \mathcal{E}$ is an instance of $\top$ and C, and it has exactly one r-successor which is an instance of D and $\top$, we would expect to see a partial context graph as shown in Figure 4.4.

$$\begin{tikzcd}
 & {[\top]} \arrow[dl] \arrow[dr] & & & {[C]} \arrow[dl] \arrow[dr] & \\
{[\top \sqcap \exists r.(\circ_1), \top]} & & {[\top \sqcap \exists r.(\circ_1), D]} & {[C \sqcap \exists r.(\circ_1), \top]} & & {[C \sqcap \exists r.(\circ_1), D]}
\end{tikzcd}$$

Figure 4.4: A partial *context graph* under construction for role r from a set of examples.

To construct the various vertices and edges of the context graph, instance chains are computed along r-successors for every subexpression corresponding to a vertex until there are either no more r-successors to expand, or a cycle is detected in the instance chain. Specifically, the set of all contexts generating new vertices from an individual i along role r with successor set $S = succ(i,r)$ is computed with the function $con(i,r,S)$ as follows:

$$con(i,r,S) \quad = con_{\exists}(i,r,S) \cup con_{\forall}(i,r,S) \cup con_{\geq}(i,r,S) \cup con_{\leq}(i,r,S)$$

where

$$\begin{aligned}
con_{\exists}(i,r,S) = \;& \{[C \sqcap \exists r.(\circ_1), D] \mid \forall C \in sc(i),\\
& \quad \forall D \in \{D \mid \forall (D,n) \in rf(S) \text{ s.t. } n > 0\}\}\\
con_{\forall}(i,r,S) = \;& \{[C \sqcap \exists r.(\circ_1), D] \mid \forall C \in sc(i),\\
& \quad \forall D \in \{D \mid \forall (D,n) \in rf(S) \text{ s.t. } n = |S|\}\}\\
con_{\geq}(i,r,S) = \;& con_{\exists}(i,r,S) \cup\\
& \{[C \sqcap {}^{\geqslant q}r.(\circ_1), A] \mid \forall C \in i_C, \forall q \text{ where } 2 \leq q \leq |S|,\\
& \quad \forall A \in \{A \mid \forall (A,n) \in rf(S) \text{ s.t. } n \geq q\}\}\\
con_{\leq}(i,r,S) = \;& con_{\forall}(i,r,S) \cup\\
& \{[C \sqcap {}^{\leqslant q}r.(\circ_1), A] \mid \forall C \in i_C, \forall q \text{ where } 1 \leq q \leq succ_{max}(C,r) - 1,\\
& \quad \forall A \in \{A \mid \forall (A,n) \in rf(S) \text{ s.t. } q \geq |S|\}\}
\end{aligned}$$

where $succ_{max}(C,r) = max\,\{n \mid \forall i \in C^{\mathcal{J}_\lambda}\ s.t.\ n = |succ(i,r)|\}$ is the maximum cardinality of r-successor sets for any instance i of $C^{\mathcal{J}_\lambda}$. However, if at the point of computing $con_{\leq}(i,r,S)$ the set $C^{\mathcal{J}_\lambda}$ is unknown for any context λ, the broader interpretation $C^{(\mathcal{I},\mathcal{U})}$ may be used until $C^{\mathcal{J}_\lambda}$ is computed and elements of $con_{\leq}(i,r,S)$ may be pruned, see Section 4.2.3 for more details. Note that we produce contexts for ${}^{\geqslant q}r.(D)$ for $q \geq 2$ because ${}^{\geqslant 1}r.(D) \equiv \exists r.(D)$, and ${}^{\leqslant q}r.(D)$ for $q \geq 1$ because ${}^{\leqslant 0}r.(D) \equiv \neg\exists r.(D) \equiv \forall r.(\neg D)$, and so are already represented by all cases in the sets $con_{\exists}(i,r,S)$ and $con_{\forall}(i,r,S)$.

Depending on the desired expressivity of the concept language, we may omit any set $con_{\diamond}(i,r,S)$ for role quantifier $\diamond \in \{\forall, \geq, \leq\}$ from the definition of $con(i,r,S)$, but will at least ensure that it contains $con_{\exists}(i,r,S)$ as a minimum.

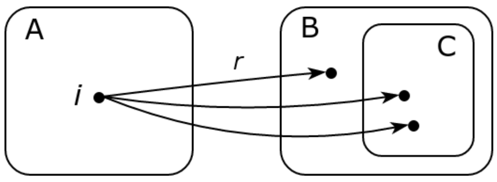


Figure 4.5: An example set of concepts and their subsumption relationships with a single individual i with three r-successors.

Example 4.2.15. *Consider the concept names $N_C = \{A, B, C\}$ with their subsumption relationships as depicted in Figure 4.5 along with the predecessor individual i with three r-successors $S = succ(i,r)$ as illustrated. In analysing the these r-successors of i, we find the* *following:*

$$\begin{aligned} |S| &= 3 \\ sc(\{i\}) &= \{A\} \cup \{\neg B, \neg C\} \\ sc(S) &= \{B, C, \neg A, \neg C\} \\ rf(S) &= \{(B,3), (C,2), (\neg A,3), (\neg C,1)\} \end{aligned}$$

The set of applicable contexts defined by $con_{\diamond}(i,r,S)$ for $\diamond \in \{\exists, \forall, \geq, \leq\}$ are as follows:

$$\begin{aligned} con_{\exists}(i,r,S) = \; & \{[A \sqcap \exists r.(\circ_1), B], [A \sqcap \exists r.(\circ_1), C], [A \sqcap \exists r.(\circ_1), \neg A], \\ & [A \sqcap \exists r.(\circ_1), \neg C], [\neg B \sqcap \exists r.(\circ_1), B], [\neg B \sqcap \exists r.(\circ_1), C], \\ & [\neg B \sqcap \exists r.(\circ_1), \neg A], [\neg B \sqcap \exists r.(\circ_1), \neg C], [\neg C \sqcap \exists r.(\circ_1), B], \\ & [\neg C \sqcap \exists r.(\circ_1), C], [\neg C \sqcap \exists r.(\circ_1), \neg A], [\neg C \sqcap \exists r.(\circ_1), \neg C]\} \\ con_{\forall}(i,r,S) = \; & \{[A \sqcap \forall r.(\circ_1), B], [A \sqcap \forall r.(\circ_1), \neg A], [\neg B \sqcap \forall r.(\circ_1), B], \\ & [\neg B \sqcap \forall r.(\circ_1), \neg A], [\neg C \sqcap \forall r.(\circ_1), B], [\neg C \sqcap \forall r.(\circ_1), \neg A]\} \\ con_{\geq}(i,r,S) = \; & \{[A \sqcap \geqslant^2 r.(\circ_1), B], [A \sqcap \geqslant^2 r.(\circ_1), C], [A \sqcap \geqslant^2 r.(\circ_1), \neg A], \\ & [\neg B \sqcap \geqslant^2 r.(\circ_1), B], [\neg B \sqcap \geqslant^2 r.(\circ_1), C], [\neg B \sqcap \geqslant^2 r.(\circ_1), \neg A], \\ & [\neg C \sqcap \geqslant^2 r.(\circ_1), B], [\neg C \sqcap \geqslant^2 r.(\circ_1), C], [\neg C \sqcap \geqslant^2 r.(\circ_1), \neg A], \\ & [A \sqcap \geqslant^3 r.(\circ_1), B], [A \sqcap \geqslant^3 r.(\circ_1), \neg A], [\neg B \sqcap \geqslant^3 r.(\circ_1), B], \\ & [\neg B \sqcap \geqslant^3 r.(\circ_1), \neg A], [\neg C \sqcap \geqslant^3 r.(\circ_1), B], [\neg C \sqcap \geqslant^3 r.(\circ_1), \neg A]\} \\ con_{\leq}(i,r,S) = \; & \{[A \sqcap \leqslant^2 r.(\circ_1), A], [A \sqcap \leqslant^2 r.(\circ_1), C], [A \sqcap \leqslant^2 r.(\circ_1), \neg B], \\ & [A \sqcap \leqslant^2 r.(\circ_1), \neg C], [\neg B \sqcap \leqslant^2 r.(\circ_1), A], [\neg B \sqcap \leqslant^2 r.(\circ_1), C], \\ & [\neg B \sqcap \leqslant^2 r.(\circ_1), \neg B], [\neg B \sqcap \leqslant^2 r.(\circ_1), \neg C], [\neg C \sqcap \leqslant^2 r.(\circ_1), A], \\ & [\neg C \sqcap \leqslant^2 r.(\circ_1), C], [\neg C \sqcap \leqslant^2 r.(\circ_1), \neg B], [\neg C \sqcap \leqslant^2 r.(\circ_1), \neg C], \\ & [A \sqcap \leqslant^1 r.(\circ_1), A], [A \sqcap \leqslant^1 r.(\circ_1), \neg B], [A \sqcap \leqslant^1 r.(\circ_1), \neg C], \\ & [\neg B \sqcap \leqslant^1 r.(\circ_1), A], [\neg B \sqcap \leqslant^1 r.(\circ_1), \neg B], [\neg B \sqcap \leqslant^1 r.(\circ_1), \neg C], \\ & [\neg C \sqcap \leqslant^1 r.(\circ_1), A], [\neg C \sqcap \leqslant^1 r.(\circ_1), \neg B], [\neg C \sqcap \leqslant^1 r.(\circ_1), \neg C]\} \end{aligned}$$

Any concept $E = nest(\lambda)$ where $\lambda \in con(i,r,S)$ will always be satisfiable with respect to $(\mathcal{I},\mathcal{U})$ because each λ was constructed over the role successors of some individual i to ensure, at the very least, that $i \in E^{(\mathcal{I},\mathcal{U})}$.

Example 4.2.15 demonstrates how the analysis of a single set S of r-successors for any individual i and role r can be used as the basis for determining the set of contexts $con_{\diamond}$ which can be used to construct new vertices of a context graph. As instance chains are computed to new individuals such as i from any vertex, $con_{\diamond}$ gives rise to new vertices which describe satisfiable subexpressions. These new vertices are then connected to their predecessors with edges, which are in turn attributed with the set of role tuples encountered by expanding r-successors. Algorithm 4 is an instance chase procedure for constructing a context graph in this way, which begins by expanding r-successors to construct instance chains to every individual and literal reachable from every example in $\mathcal{E}$. Each individual reached in the chase is analysed with respect to each role name $r \in N_R$ to construct a new set of role subexpression contexts λ which comprise the vertices of the context graph.

Algorithm 4 Instance chase procedure for computing context graph $\mathfrak{G}$ for a limited number of restricted contexts λ.

```
1:  $\mathfrak{G} = (V, E)$ where $V = \emptyset$, $E = \emptyset$            ▷ Initialise an empty context graph
2:  $Q := \{(e, [], e) \mid \forall e \in \mathcal{E}\}$      ▷ Example $e$ with preceding instance chain $\mathfrak{ic} = []$
3:  $L := \emptyset$   ▷ Containing tuples $(\lambda, i, e)$ to record $i$ as reachable from $e$ in context $\lambda$
4:  while $Q \neq \emptyset$ do
5:      $(i, \mathfrak{ic}, e) \in Q$                              ▷ Select an arbitrary tuple, then...
6:      $Q := Q \setminus \{(i, \mathfrak{ic}, e)\}$                               ▷ ...remove it from $Q$
7:      if $i_\lambda = \emptyset$ then                       ▷ $i_\lambda$ holds all preceding contexts for $i$
8:          $i_\lambda := \{[A] \mid A \in sc(i)\}$
9:      end if
10:     $succ_i = \emptyset$                                ▷ All successors of $i$ for any role
11:     for all $r \in N_R$ do
12:         $S = \{j \mid \langle i, j \rangle \in r^{(\mathcal{I},\mathcal{U})}\}$                         ▷ All $r$-successors of $i$
13:         $succ_i := succ_i \cup S$
14:         for all $[C_1 \sqcap \Diamond r.(\circ_1), C_2] \in con(i, r, S)$ do
15:             for all $\lambda \in i_L$ where $\lambda = [D_1, \ldots, D_n]$ and $D_n = C_1$ do       ▷ $(n \geq 1)$
16:                 $\lambda' := [D_1, \ldots, D_n \sqcap \Diamond r.(\circ_n), C_2]$
17:                 $V := V \cup \{\lambda, \lambda'\}$                          ▷ Add graph vertices
18:                 $E := E \cup \{(\lambda, \lambda')\}$                            ▷ Add graph edge
19:                 $(\Delta_\lambda := \Delta_\lambda \cup \{i\})$
20:                 $L := L \cup \{(\lambda, j, e)\}$         ▷ Label $j$ as reachable from example $e$ in $\lambda$
21:                 for all $j \in S$ where $j \in C_2^{(\mathcal{I},\mathcal{U})}$ do
22:                     $\Delta_{\lambda'} := \Delta_{\lambda'} \cup \{j\}$
23:                     $j_\lambda := j_L \cup \{\lambda'\}$              ▷ $j_\lambda$ holds all preceding contexts for $j$
24:                 end for
25:             end for
26:         end for
27:     end for
28:     for all $j \in succ_i$ where $j$ not contained in $\mathfrak{ic}$ do                  ▷ Prevent cycles
29:         $\mathfrak{ic}' := \mathfrak{ic} \| [\langle i, j \rangle)]$          ▷ Update instance chain with new role successor $j$
30:         $Q := Q \cup \{(j, \mathfrak{ic}', e)\}$                    ▷ Add new tuple to chase from $j$
31:     end for
32: end while
```

The instance chase Algorithm 4 will always terminate as it enumerates all acyclic instance chains from the finite set of example instances $\mathcal{E}$ over $(\mathcal{I}, \mathcal{U})$. After construction of the context graph $\mathfrak{G}$ over a knowledge base $\mathcal{K}$ for a set of examples $\mathcal{E}$, there will be n contexts λ_k where $1 \leq k \leq n$.

Initially, the context graph is potentially large for knowledge bases with many concept and role names which describe the individual and literal data reachable from examples in $(\mathcal{I},\mathcal{U})$. As the purpose of the context graph is to enable a refinement operator to select appropriate refinements of concept expressions in any subexpression context, we can make several observations about how it will be used, and how it can be limited in size. After construction by the chase Algorithm 4, the context graph will contain a set of vertices corresponding to subexpression contexts λ and with each, a local domain Δ_λ capturing all individuals or literals known to reside in each context. Given each local domain consisting of individuals, we can compute the set of local equivalence, subsumption and disjointness axioms $\mathcal{T}_\lambda$ between pairs of simple atomic and negated atomic concept names.

A refinement operator may utilise the axioms of $\mathcal{T}_\lambda$ generated by analysing each local domain Δ_λ for each context λ to ensure that refinements avoid expressions which are clearly unsatisfiable, or steps which are clearly improper or redundant. For example, concept equivalence, subsumption and disjointness between concepts C, D can be utilised to ensure that a conjunction $C \sqcap D$ is never produced in the context of λ. After the analysis of each Δ_λ in the context graph, we identify the groups of concepts which are found to be equivalent, and construct so-called *local equivalence groups* as follows.

Definition 4.2.16. ***(Local Equivalence Group)*** *A* ***local equivalence group*** *relative to a context λ is a set of equivalent concepts $\{C_1, \ldots, C_k \mid C_i \equiv C_j \in \mathcal{T}_\lambda (1 \leq i < j \leq k)\}$.*

For any local equivalence group G of any $\mathcal{T}_\lambda$, we identify a single concept name $C \in G$ to represent all concepts in G for the purposes of refinement, as permitting refinement to concepts from $G \setminus \{C\}$ would produce redundant refinement chains, such as where $G = \{A, B\}$ and the two chains $C \rightsquigarrow C \sqcap A$ and $C \rightsquigarrow C \sqcap B$ where $A \equiv B$ in the subexpression context λ. In order to select an appropriate concept, we may rely on user defined preference, or simply choose the most general concept of G relative to the TBox $\mathcal{T}$ as our intention is to use the refinement operator for generating concepts as hypotheses, and aim to ensure hypotheses generalise well over unseen data, which is discussed in more detail in Chapter 5.

Generally, any concept expression interpreted relative to a context λ is considered *locally minimal* if it is not equivalent under $\mathcal{J}_\lambda$ to some other concept as follows.

Definition 4.2.17. ***(Locally Minimal Concept Expression)*** *A concept expression C is* ***locally minimal*** *in context λ under interpretation $\mathcal{J}_\lambda$ iff, for all subexpressions $S \in subex(C)$ where $S = S_1 \sqcap \ldots \sqcap S_n$ or $S = S_1 \sqcup \ldots \sqcup S_n$, that removing any subexpression S_i where $1 \leq i \leq n$ resulting in concept C' we find that $C \not\equiv_{\mathcal{J}_\lambda} C'$.*

4.2.3 Pruning the Context Graph

The local axioms $\mathcal{T}_\lambda$ for each context λ give rise to ways of *pruning* the context graph to exclude irrelevant portions. To explain how, consider the following fragment of a context graph:

$$\begin{tikzcd} & \lambda_1 = [\ldots, A] \arrow[dl] \arrow[d] \arrow[dr] & \\ \lambda_2 = [\ldots, A \sqcap \exists r.(\circ), \top] & \lambda_3 = [\ldots, A \sqcap \exists r.(\circ), B] & \lambda_4 = [\ldots, A \sqcap \exists r.(\circ), \neg A] \end{tikzcd}$$

In this case, all r-successors of individuals in $A^{\mathcal{J}_{\lambda_1}}$ for $\exists r.(\circ)$ were instances of $\top$, B and $\neg A$. As no r-successor was an instance of A, we would find that $\top \equiv_{\mathcal{J}_\lambda} \neg A$ in each context λ_2, λ_3 and λ_4. As we would otherwise opt for the more general concept $\top$ over $\neg A$ in refinement within these contexts, we may safely prune edge (λ_1, λ_4) in preference over (λ_1, λ_2), and along with it the entire subtree rooted at vertex λ_4.

In general, consider any context graph $\mathfrak{G} = (V, E)$ and any vertex $\lambda \in V$ where $\lambda = [\ldots, C]$ with edge $(\lambda, \lambda') \in E$ where $\lambda' = [\ldots, C \sqcap \Diamond r.(\circ), D]$. For any local equivalence group eq over local axioms $\mathcal{T}_{\lambda'}$, we may select a most general concept $M \in eq$ and exclude all others from the context graph along any edge $(\lambda, \lambda'') \in E$ where $\lambda'' = [\ldots, C \sqcap \Diamond r.(\circ), F]$ for $F \in eq \setminus \{M\}$ where $M \equiv F \in \mathcal{T}_{\lambda'}$ such that $M^{\mathcal{J}_{\lambda'}} = F^{\mathcal{J}_{\lambda'}}$. Pruning the vertex λ'' from the graph removes the edge (λ, λ'') and the subtree under λ'' from $\mathfrak{G}$. In doing so, we are eliminating portions of the context graph which describe concepts which a refinement operator never ought to consider, because if $M \equiv_{\mathcal{J}_\lambda} F$ for any concept F in context λ'', then any concept composed with F is equivalent to concepts where F has been replaced with M, otherwise permitting redundant refinement steps. Furthermore, any concept containing $F \sqcap M$ or $F \sqcup M$ would not be locally minimal, also potentially a source of redundant or improper refinement steps.

Recall that in the construction of contexts $\lambda = [\ldots, {\leqslant^n} r.(\circ), C]$ which were computed by $con_{\leq}(i, r, S)$ from some individual i and r-successors S, that the maximum value for n was initially determined by the maximum size $|S|$ for any $i \in N_I$ across the entire knowledge-base. However, after the context graph has been computed, the set of instances in each local domain Δ_λ permit us to compute tighter upper bounds for n by inspecting the maximum size S for individuals in the local domain Δ_λ only. Afterwards, we may find that if the maximum size of any set of r-successors which are instances of C is m, then any vertex such as $\lambda = [\ldots, {\leqslant^n} r.(\circ), C]$ where $n > m$ may also be pruned from the context graph, along with the subtree rooted at any pruned vertices.

4.2.4 Identifying Concept Subexpression Contexts

Computing the context-specific interpretation $D^{\mathcal{J}_\lambda}$ of some arbitrary subexpression D belonging to some expression C, we necessarily require the local domain Δ_λ of the context λ describing where D is situated within C. However, as we have seen in the previous section, we do not propose to compute all possible contexts λ representing all possible subexpressions of C, as this is practically infeasible. Instead, we employ a context graph to capture a subset of subexpression contexts, which may not be associated with all local domains.

In this section, we describe how a precise subexpression context λ which identifies any subexpression D in C, along with the local domain Δ_λ, can be approximated over multiple contexts $\lambda_1, \ldots, \lambda_n$ capturing simpler and more general concepts within a context graph. To begin, we distinguish exact subexpression contexts from approximate ones with an example.

Example 4.2.18. *Consider the following concept expression:*

$$C = \overbrace{A \sqcap B \sqcap \exists r.(\underbrace{D \sqcup (\forall s.(\underbrace{E \sqcap F}_{\lambda_3})))}_{\lambda_2}}^{\lambda_1}$$

In this expression, the following three exact role subexpression contexts apply:

$$\begin{aligned}
\lambda_1 &= [A \sqcap B \sqcap \exists r.(D \sqcup (\forall s.(E \sqcap F)))] \\
\lambda_2 &= [A \sqcap B \sqcap \exists r.(\circ_1), D \sqcup (\forall s.(E \sqcap F))] \\
\lambda_3 &= [A \sqcap B \sqcap \exists r.(\circ_1), D \sqcup (\forall s.(\circ_2)), E \sqcap F]
\end{aligned}$$

As the exact contexts λ_1, λ_2 and λ_3 from Example 4.2.18 contain a disjunction ($\sqcup$), they cannot be represented in a context graph because the vertices are restricted to capturing subexpressions of the form $[C, \ldots, \Diamond r.(\circ), D]$ for simple concepts C, D by

Definition 4.2.12. Therefore, the local domain Δ_λ associated with any vertex λ in the context graph will not match the local domain of λ_1, λ_2 and λ_3. However, a context graph may contain contexts representing subexpressions which are super-classes of some arbitrary subexpression D. In this case, we may still derive useful information about the local domain of D, along with any local axioms about the local domain. As an example, consider a context graph $\mathfrak{G}$ which contains the following vertices and edges representing subexpression contexts λ_i for $4 \leq i \leq 14$ as shown below:

$$\begin{aligned}
\lambda_4 &= [\top] \\
\lambda_5 &= [A] \\
\lambda_6 &= [B] \\
\lambda_7 &= [A \sqcap \exists r.(\circ_1), \top] \\
\lambda_8 &= [A \sqcap \exists r.(\circ_1), D] \\
\lambda_9 &= [B \sqcap \exists r.(\circ_1), \top] \\
\lambda_{10} &= [B \sqcap \exists r.(\circ_1), D] \\
\lambda_{11} &= [A \sqcap \exists r.(\circ_1), \top \sqcap \forall s.(\circ_2), E] \\
\lambda_{12} &= [A \sqcap \exists r.(\circ_1), \top \sqcap \forall s.(\circ_2), F] \\
\lambda_{13} &= [B \sqcap \exists r.(\circ_1), \top \sqcap \forall s.(\circ_2), E] \\
\lambda_{14} &= [B \sqcap \exists r.(\circ_1), \top \sqcap \forall s.(\circ_2), F]
\end{aligned}$$

$$\begin{tikzcd}
\lambda_4 & & \lambda_5 \arrow[dl] \arrow[dr] & & & \lambda_6 \arrow[dl] \arrow[dr] & \\
& \lambda_7 \arrow[dl] \arrow[dr] & & \lambda_8 & \lambda_9 \arrow[dl] \arrow[dr] & & \lambda_{10} \\
\lambda_{11} & & \lambda_{12} & \lambda_{13} & & \lambda_{14} &
\end{tikzcd}$$

Continuing on from Example 4.2.18, consider again the concept $C = A \sqcap B \sqcap \exists r.(D \sqcup (\forall s.(E \sqcap F)))$ and the context λ_1 capturing this whole expression. From the context graph $\mathfrak{G}$ above, we find that the subexpression contexts λ_4, λ_5 and λ_6 all represent subexpressions which are super-classes of C, as $C \sqsubseteq \top$, $C \sqsubseteq A$, and $C \sqsubseteq B$. This also means that the local domain $\Delta_{\lambda_1} \subseteq \Delta_{\lambda_i}$ where $4 \leq i \leq 6$. In fact, as each of these local domains contains only those individuals which are instances of their respective subexpression concepts, we will find that $\Delta_{\lambda_1} \subseteq \bigcap_{4 \leq i \leq 6} \Delta_{\lambda_i}$, the intersection of each domain, because $C = A \sqcap B \sqcap \ldots$ or the intersection of these subexpressions[1]. As we will soon show, any local axioms which hold in any of the contexts λ_4, λ_5 or λ_6 will apply to λ_1, and therefore be applicable for use by a refinement operator when modifying the concept in context λ_1.

Further continuing on from Example 4.2.18, consider the context $\lambda_2 = [A \sqcap B \sqcap \exists r.(\circ_1), D \sqcup (\forall s.(E \sqcap F))]$ which refers to the subexpression $D \sqcup (\forall s.(E \sqcap F))$. There are also a number of subexpression contexts in the context graph $\mathfrak{G}$ above which describe super-classes of this subexpression, namely λ_7 and λ_9, as both these contexts

[1]Note that $A \sqcap B \equiv A \sqcap B \sqcap \top$.

refer to the subexpression $\top$, and each preceding subexpression is a superclass of C where $C \subseteq A \sqcap \exists r.(\top)$ and $C \subseteq B \sqcap \exists r.(\top)$. Similarly, the context $\lambda_3 = [A \sqcap B \sqcap \exists r.(\circ_1), D \sqcup (\forall s.(\circ_2)), E \sqcap F]$ which refers to the subexpression $E \sqcap F$ is a subclass of the subexpressions referred to by λ_{11}, λ_{12}, λ_{13} and λ_{14}.

Generally, given any arbitrary subexpression D and role subexpression context λ, there may be multiple subexpression contexts $\bar{\lambda} = \{\lambda_1, \ldots, \lambda_n\}$ from a context graph which, when considered together, approximate λ. Formally, we denote the set of such contexts $\bar{\lambda}$ as *applicable subexpression contexts* relative to an exact context λ for some arbitrarily complex subexpression D.

Definition 4.2.19. ***(Applicable Subexpression Contexts)*** *Any context $\lambda' = [D_1, \ldots, D_m]$ is **applicable** to describe a subexpression S of a concept C which is otherwise identified by an exact context $\lambda = [C_1, \ldots, C_m]$ iff $C_i \sqsubseteq_{(\mathcal{I},\mathcal{U})} D_i$ for $1 \leq i \leq n$, defined as $appl(C, \lambda, S)$ where:*

$$
\begin{aligned}
&appl(C, [D_1], S) \leftarrow C = S \wedge S \sqsubseteq_{(\mathcal{I},\mathcal{U})} D_1\\
&appl(\Diamond r.(C), [D_1 \sqcap \Diamond r.(\circ_1), D_2, \ldots, D_n], S) \leftarrow D_1 = \top \wedge\\
&\qquad appl(C, [D_2, \ldots, D_n], S) \ (for\ n \geq 2)\\
&appl(C_1 \sqcap \ldots \sqcap C_m \sqcap \Diamond r.(C), [D_1 \sqcap \Diamond r.(\circ_1), D_2, \ldots, D_n], S) \leftarrow\\
&\qquad \exists C_i \ s.t.\ C_i \sqsubseteq_{(\mathcal{I},\mathcal{U})} D_1 \wedge appl(C, [D_2, \ldots, D_n], S) \ (for\ 1 \leq i \leq m, m \geq 1, n \geq 2)\\
&appl(C_1 \sqcup \ldots \sqcup C_m, [D_1 \sqcap \Diamond r.(\circ_1), D_2, \ldots, D_n], S) \leftarrow C_1 \sqcup \ldots \sqcup C_m \sqsubseteq_{(\mathcal{I},\mathcal{U})} D_1 \wedge\\
&\qquad \exists C_i \ s.t.\ appl(C_i, [D_2, \ldots, D_n], S) \ (for\ 1 \leq i \leq m, m \geq 2, n \geq 2)
\end{aligned}
$$

*From a set of contexts $V = \{\lambda_1, \ldots, \lambda_n\}$ for $n \geq 1$, the set of contexts which are **most applicable** to describe subexpression S of a concept C is the set $\bar{\lambda}$ where:*

$$
\begin{aligned}
\bar{\lambda} \ &= \{[D_1, \ldots, D_n] \mid \forall [D_1, \ldots, D_n] \in V \ for\ n \geq 1 \wedge appl(C, \lambda, S) \wedge\\
&\qquad \neg \exists [D'_1, \ldots, D'_n] \in V \ s.t.\ D'_i \sqsubset_{(\mathcal{I},\mathcal{U})} D_i \ for\ any\ 1 \leq i \leq n\}
\end{aligned}
$$

Intuitively, $\bar{\lambda}$ is the set of contexts λ which most tightly describe a subexpression context S.

While this method only generally provides us with an *estimate* of the exact role subexpression context λ for any concept expression interpreted by $\mathcal{J}_\lambda$, it can still be used to detect when subexpressions may be unsatisfiable or not locally minimal, as we are motivated to detect. For example, once again consider the concept $C = A \sqcap B \sqcap \exists r.(D \sqcup (\forall s.(E \sqcap F)))$ where $\bar{\lambda} = \{\lambda \mid \forall \lambda \in \{\lambda_4, \ldots, \lambda_{14}\} \rightarrow appl(C, \lambda, C)\} = \{\lambda_5, \lambda_6\}$. With respect to these contexts, if we find that $A \sqcap B \sqsubseteq \bot \in \mathcal{T}_{\lambda_5}$ or $A \sqcap B \sqsubseteq \bot \in \mathcal{T}_{\lambda_6}$ then the subexpression $A \sqcap B$ is unsatisfiable with respect to its exact subexpression context λ.

Proposition 4.2.20. *For any subexpression S of a concept C and any applicable context λ' for S where $appl(C, \lambda', S)$, it is always the case that $\forall \phi$ where $\mathcal{T}_{\lambda'} \models \phi$ then $\mathcal{T}_{\lambda} \models \phi$ where λ is the exact subexpression context for S.*

Proof. By Proposition 4.2.11, we know that if $\mathcal{T} \models \phi$ holds over the domain $\Delta^{(\mathcal{I},\mathcal{U})}$ for any axiom ϕ defined for local axioms in Definition 4.2.9, then $\mathcal{T}_{\lambda} \models \phi$ holds over any local domain Δ_{λ} when $\Delta_{\lambda} \subseteq \Delta^{(\mathcal{I},\mathcal{U})}$. Therefore, for Proposition 4.2.20 to hold, we need to show that $\Delta_{\lambda} \subseteq \Delta_{\lambda'}$ always holds. For the exact context $\lambda = [D_1, \ldots, D_m]$ and any applicable context $\lambda' = [D'_1, \ldots, D'_m]$ for S, it must have been the case that each $D_i \sqsubseteq_{(\mathcal{I},\mathcal{U})} D'_i$ for $1 \leq i \leq m$ by Definition 4.2.19. Consider the case where $\lambda' = [D'_1]$ and $\lambda = [D_1]$, and where both $D'_1 = \top$ and $D_1 = \top$. Then, as $D_1 \sqsubseteq_{(\mathcal{I},\mathcal{U})} D'_1$, we have $\Delta_{\lambda} \subseteq \Delta'_{\lambda}$ as required by the definition of $\top$ relative to each local domain. Now consider the case where $\lambda' = [D'_1, \ldots, D'_m]$ and $\lambda = [D_1, \ldots, D_m]$ for $m \geq 2$, where each $D'_i = C'_i \sqcap \Diamond r.(\circ)$ for $1 \leq i < m$. We know that D_i has the form $\Diamond r.(\circ)$ or $C_1 \sqcap \ldots C_k \sqcap \Diamond r.(\circ)$, or $C_1 \sqcup \ldots C_k \sqcup \Diamond r.(\circ)$ for $k \geq 1$ describing the r-successors $\circ$ which are instances of the concept $nest([D_{i+1}, \ldots, D_m])$. Because λ' is applicable, it must have been the case that $nest([D_i, \ldots, D_m]) \sqsubseteq_{(\mathcal{I},\mathcal{U})} C'_i$. Therefore, the set of instances of C'_i with r-successors is a superset of the set of instances of $nest([D_i, \ldots, D_m])$ with r-successors for any role quantifier $\Diamond$. Therefore, the local domain $\Delta_{\lambda'_{i+1}}$ for the context $\lambda'_{i+1} = [D'_1, \ldots, C'_i \sqcap \Diamond r.(\circ), nest([D'_{i+1}, \ldots, D'_m])]$ for the subexpression at level $i+1$ will always be a superset of the local domain Δ_{λ_i} for the exact context $\lambda_i = [D_1, \ldots, D_i, nest([D_{i+1}, \ldots, D_m])]$, which shows that $\Delta_{\lambda} \subseteq \Delta_{\lambda'}$ always holds. □

Corollary 4.2.21. *From the proof of Proposition 4.2.11, we know that each context λ' in the set of most applicable contexts $\lambda' \in \bar{\lambda}$ also each have $\Delta_{\lambda} \subseteq \Delta'_{\lambda}$. Therefore, the local domain of the exact context Δ_{λ} must also be a subset of the intersection of the local domains for all most applicable contexts $\bar{\lambda}$ as $\Delta_{\lambda} \subseteq \Delta_{\lambda_1} \cap \ldots \cap \Delta_{\lambda_n}$ for all $\lambda_i \in \bar{\lambda}$ for $1 \leq i \leq n$.*

Proposition 4.2.22. *Consider a subexpression $S = C_1 \sqcap \ldots \sqcap C_n$ for $n \geq 2$ and pair C_i, C_j for $1 \leq i < j \leq n$ with a set of applicable subexpression contexts $\bar{\lambda}$. If, for any context $\lambda' \in \bar{\lambda}$ where $\lambda' = [D'_1, \ldots, D'_m]$ for $m \geq 1$ where $D_m = C_i$ we have $\mathcal{T}_{\lambda'} \models C_i \sqcap C_j \sqsubseteq \bot$ then S is unsatisfiable relative to $\mathcal{J}_{\lambda}$ where λ is the exact role subexpression context of S.*

Proof. From the proof of Proposition 4.2.20, we know that the local domain Δ'_{λ} subsumes the local domain Δ_{λ} for the exact context $\lambda = [D_1, \ldots, D_m]$ where $S = D_m$. Therefore, if $\mathcal{T}_{\lambda'} \models C_i \sqcap C_j \sqsubseteq \bot$, then it must be the case that $\mathcal{T}_{\lambda} \models C_i \sqcap C_j \sqsubseteq \bot$. □

Proposition 4.2.23. *Consider a subexpression S and the set of all applicable subexpression contexts $\bar{\lambda}$. If, for all $\lambda' \in \bar{\lambda}$ where $\lambda' = [D_1, \ldots, D_m]$ for $m \geq 1$ the context graph does not contain a vertex $\lambda'' = [D_1, \ldots, D_m \sqcap \Diamond r.(\circ), D_{m+1}]$, then the concept expression $T = S \sqcap \Diamond r.(D_{m+1})$ must be unsatisfiable relative to the exact role subexpression context λ where $T^{\mathcal{J}_\lambda} = \emptyset$.*

Proof. From the proof of Proposition 4.2.20, we know that the local domain Δ'_λ subsumes the local domain Δ_λ for the exact context $\lambda = [D_1, \ldots, D_m]$ where $S = D_m$. Therefore, if there were no individuals in $\Delta_{\lambda'}$ which are also instances of $D_m \sqcap \Diamond r.(D_{m+1})$, then the context graph would not contain any edge (λ', λ''). Therefore, if the edge $(\lambda', \lambda'') \notin E$, we find that the expression T must be unsatisfiable as it is interpreted over the local domain Δ_λ which is subsumed by Δ'_λ. □

Each vertex in a context graph $\mathfrak{G}$ labelled with context $\lambda = [D_1, \ldots, D_m]$ where each D_i for $1 \leq i < m$ has the form $C_i \sqcap \Diamond r.(\circ)$ represents a satisfiable expression $nest(\lambda)$. This is because each context λ was generated from role successors along instance chains inspected in $(\mathcal{I}, \mathcal{U})$ which capture the instances of each subexpression referred to by $\lambda_i = [D_1, \ldots, D_i \sqcap \Diamond r.(\circ), nest([D_{i+1}, \ldots, D_m])]$ for $1 \leq i < m$. Continuing our example, the local domains Δ^{λ_5} and Δ^{λ_6} capture all those instances of A and B in their respective contexts. Note that the local domain Δ^{λ_4} where $\lambda_4 = [\top]$ is a superset of both of these where $\Delta^{\lambda_5} \subseteq \Delta^{\lambda_4}$ and $\Delta^{\lambda_6} \subseteq \Delta^{\lambda_4}$. Therefore, if $A \sqcap B \sqsubseteq \bot \in \mathcal{T}_{\lambda_4}$, then it will also hold in $\mathcal{T}_{\lambda_5}$ and $\mathcal{T}_{\lambda_6}$.

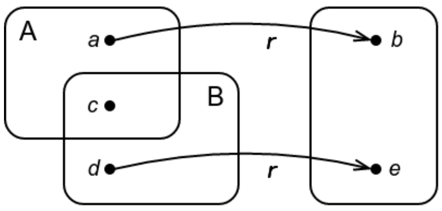


Figure 4.6: Concepts A, B have various instances a, c, d and r-successors b, e. While $A \sqcap B$ is satisfiable, no instance of $A \sqcap B$ has an r-successor.

Even if $A \sqcap B$ is determined to be satisfiable in contexts λ_4, λ_5 and λ_6, we cannot determine from the context graph that $A \sqcap B \sqcap \exists r.(\top)$ is satisfiable from the contexts λ_7 and λ_9. To illustrate why, consider Figure 4.6 where $A \sqcap B$ is satisfiable, but where the only instance c does not have an r-successor. In this case, contexts λ_7 and λ_9 would still have been generated by the instance chase over the role tuples $\langle a, b \rangle$ and $\langle d, e \rangle$, therefore their existence does not necessarily indicate the satisfiability of $A \sqcap B \sqcap \exists r.(\top)$. Consider the case where we exclude tuple $\langle d, e \rangle$, and where only instances of A had r-successors. Then, λ_9 would not exist in the context graph, as no instance of B has an r-successor. Therefore, we can identify when $A \sqcap B \sqcap \exists r.(\top)$ is *clearly unsatisfiable* when the context graph does not contain vertices of the form $[\phi \sqcap \exists r.(\circ), \top]$ for all $\phi \in \{A, B\}$, all the operands in the conjunction, as shown in Proposition 4.2.23.

In general, if any subexpression S of a concept expression C has an empty set of most applicable contexts $\mathcal{K}$, while it does mean that S is unsatisfiable with respect to $\mathcal{J}_\lambda$, it does not necessarily mean that C is unsatisfiable under $(\mathcal{I},\mathcal{U})$ where $C^{(\mathcal{I},\mathcal{U})} = \emptyset$. To understand why, consider the case when $C = A \sqcup D$ where $D^{\mathcal{J}_\lambda} = \emptyset$, where C is still satisfiable as A. Also, consider when $C = A \sqcap \forall r.(B)$ and where no instance i of A has any r-successors. Then, C is clearly satisfiable with respect to $(\mathcal{I},\mathcal{U})$. However, as there were no r-successors of A, the context graph would not contain any vertex labelled with λ where $\lambda = [A \sqcap \forall r.(\circ), B]$ as no r-successors of A were present to construct it. Therefore, the set of most applicable contexts for the subexpression B would be empty, as would its context specific interpretation $B^{\mathcal{J}_\lambda}$. In this case, the subexpression $\forall r.(B)$ was not particularly interesting, as it describes a constraint on data which was not present in the knowledge base, and thus is not informative. As our goal is to use the context specific interpretation and context graph to guide a refinement operator in the construction of concepts which describe data for the purposes of machine learning and data mining, we are not concerned with expressions which do not cover at least some portion of the asserted data. For this reason, we may impose the restriction that *every* subexpression S of any concept C under consideration by refinement have a non-empty set of most applicable contexts which, at the very least, indicate potential satisfiability of S and of C. We encapsulate this notion by defining the boolean function $ncu(C)$ which returns true iff all subexpressions S of C have a non-empty set of most applicable contexts $\bar{\lambda}$ and are therefore *not clearly unsatisfiable*, as follows:

$$ncu(C) \leftrightarrow \forall S \in sub(C) : \bar{\lambda} \neq \emptyset$$

We later use the function $ncu(S)$ to ensure that all subexpressions S of a refined expression C have non-empty most applicable context sets $\bar{\lambda}$ and therefore may be satisfiable, and do not clearly cover none of the data in the knowledge base.

In summary, the context graph provides a way of permitting us to attribute a set of broadly applicable contexts $\bar{\lambda}$ to any subexpression S of a concept C by retrieving all applicable subexpression contexts $appl(C, V, S)$ for the set of contexts V in a context graph $\mathfrak{G} = (V, E)$. By attributing the set of applicable contexts $\bar{\lambda}$ to any subexpression S of C, we are able to compute from each set of local axioms $\mathcal{T}_\lambda$ for $\lambda \in \bar{\lambda}$ together with the context graph whether:

- The conjunction of new atomic or negated atomic concepts with S results in an unsatisfiable or non locally minimal expression by Proposition 4.2.22;
- The conjunction of a new quantified role expression with S results in an unsatisfiable expression by Proposition 4.2.23.

In the next section, we will define a new pair of downward and upward refinement operators called $\rho_{\bar{\lambda}}$ and $v_{\bar{\lambda}}$ which operate in accordance with these restrictions by inspection of a pre-computed context graph $\mathfrak{G}$ and accompanying sets of local domains Δ_λ and local axioms $\mathcal{T}_\lambda$. It is important to note that these two restrictions will not prevent the downward operator $\rho_{\bar{\lambda}}$ from reaching any part of the concept space which may otherwise contain concepts which are solutions to a problem. This is because atomic concepts cannot themselves be refined further, so their exclusion by the identification of their unsatisfiability or improperness in conjunction with other expressions does not impede the ability of $\rho_{\bar{\lambda}}$ to reach new concepts. New quantified role expressions *can* be refined to reach other concepts which could be solutions, however not if the new quantified role expression is already unsatisfiable, as any downward refinements produced by $\rho_{\bar{\lambda}}$ from such expressions will also be unsatisfiable. Therefore, these restrictions are designed to improve efficiency but not at the cost of search completeness. Nevertheless, there are practical limitations which require us to enforce restrictions on the permissible behaviours of $\rho_{\bar{\lambda}}$ and $v_{\bar{\lambda}}$ which *do* result in these operators being incomplete overall, as we will discuss in Sections 4.3.4 and 4.5.1.

4.3 Building a Context-Specific Refinement Operator

In this section, we will introduce a new refinement operator which utilises the information captured in local axioms generated by constructing a context graph to permit it to avoid certain improper or redundant refinement steps or the construction of unsatisfiable concepts. Use of such an operator in generate-and-test learning algorithms aims to improve the efficiency of the search by excluding concepts which do not contribute towards the search for solutions. We present our refinement operator by addressing each of the refinement cases of ρ_B as described in Section 4.1 and cover how the context graph and local axioms can be used in each case.

4.3.1 Atomic and Negated Atomic Concepts

Given a downward refinement operator ρ, a conjunctive step $\overset{\sqcap}{\rightsquigarrow}$ is defined as refinement of the kind:

$$\rho(C) \overset{\sqcap}{\rightsquigarrow} C \sqcap D$$

for concept expressions C and D. For this refinement step to be proper we require that $C \not\sqsubseteq D$, for it to be non-redundant we require that $D \not\sqsubseteq C$, and for $C \sqcap D$ to be satisfiable we require $C \sqcap D \not\sqsubseteq \bot$. In this section, we consider the case when concepts C and D are both either atomic or negated atomic concepts. The definition of ρ_B includes the following relevant cases for refinement of a concept C:

$$\begin{array}{ll} \{A' \mid A' \in sh_{\downarrow}(A)\} \cup \{A \sqcap D | D \in \rho_B(\top)\} & \text{if } C = A\ (A \in N_C) \\ \{\neg A' \mid A' \in sh_{\uparrow}(A)\} \cup \{\neg A \sqcap D | D \in \rho_B(\top)\} & \text{if } C = \neg A\ (A \in N_C) \end{array}$$

These cases describe the conjunction of any atomic term with another in the context of the role range concept B. Assume the concepts $\{A, B, C, D\} \in N_C$ and apply in the context of B, and where $\{D \sqsubseteq C, C \sqsubseteq B, A \sqcap D \sqsubseteq \bot\} \subseteq \mathcal{T}$ such that $\{A, \neg A, C, \neg C, D, \neg D\} \in M_B$. Then, the rules above permit the construction of the following refinement chains:

1. $A \rightsquigarrow A \sqcap A$ (as $A \in \rho_\top(\top)$, however $A \equiv A$)
2. $A \rightsquigarrow A \sqcap \neg A$ (as $\neg A \in \rho_B(\top)$, however $A \sqcap \neg A \sqsubseteq \bot$)
3. $A \rightsquigarrow A \sqcap C$ (as $C \in \rho_\top(\top)$)

We can simply explicitly avoid the first two cases by recognising them syntactically, however we cannot do so in the last case. While $A \sqcap C$ may be satisfiable with respect to $\mathcal{T}$, as we discussed in Section 4.2 this concept relative to a particular subexpression context λ may be unsatisfiable where $\mathcal{T}_\lambda \models A \sqcap B \sqsubseteq \bot$. Because ρ_B is not aware of context-specific information about the satisfiability of such concepts, it will otherwise always permit $A \sqcap C$. Furthermore, consider the refinement case of ρ_B for handling conjunctive expressions $C_1 \sqcap \ldots \sqcap C_n$ for $n \geq 2$:

$$\{C_1 \sqcap \ldots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \ldots \sqcap C_n | D \in \rho_B(C_i), 1 \leq i \leq n\}$$

If the concept chosen to refine within the expression C_i is atomic, then the atomic refinement case above applies, but does not take into account any of the other expressions in the conjunction. In this case, such a refinement step would be permitted:

$$A \sqcap B \sqcap C \rightsquigarrow A \sqcap B \sqcap D \quad \text{(as } D \in \rho_\top(C)\text{, however } A \sqcap D \sqsubseteq \bot \in \mathcal{T}\text{)}$$

The operator ρ_B permitted refinement to the unsatisfiable concept $A \sqcap B \sqcap D$ in this case because C was refined in isolation of the remainder of the conjunctive expression and ignored the other conjuncts. To motivate a tighter definition for a refinement operator over atomic and negated concepts within any particular local context λ, consider the following example.

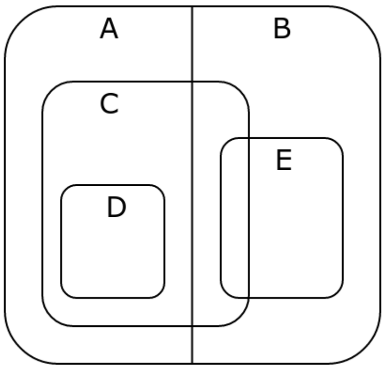


Figure 4.7: A diagram representing subsumption relationships between an example set of concept names $N_C = A, B, C, D, E$ modelled by local axioms $\mathcal{T}_\lambda$ relative to some subexpression context λ.

Example 4.3.1. *Figure 4.7 represents the entirety of a local domain Δ_λ with the following two local equivalence groups:*

$$\{B, \neg A\} \quad \{A, \neg B\}$$

For the purposes of refinement, we may exclude concepts $\neg A$ and $\neg B$ in preference for their simpler equivalent concepts B and A. The remaining pairwise relationships between the reduced set of nine concepts $\{\top, A, B, C, \neg C, D, \neg D, E, \neg E\}$ as captured by $\mathcal{T}_\lambda$ along with overlapping concepts which are not related by axioms in $\mathcal{T}_\lambda$ are as follows:

Strict subsumption	*Overlap*	*Disjoint*
$E \sqsubset B \sqsubset \neg D \sqsubset \top$	$A : C, \neg C, \neg D$	$A : B, E$
$D \sqsubset A \sqsubset \neg E \sqsubset \top$	$B : C, \neg C, \neg E$	$B : A, D$
$D \sqsubset C \sqsubset \top$	$C : A, B, E, \neg D, \neg E$	$C : \neg C$
$\neg C \sqsubset \top$	D : *(none)*	$D : B, E, \neg D$
	$E : C, \neg C$	$E : A, D, \neg E$
	$\neg C : A, B, E, \neg E$	$\neg C : C$
	$\neg D : A, C, \neg E$	$\neg D : D$
	$\neg E : B, C, \neg C, \neg D$	$\neg E : E$

The strict subsumption and disjointness cases describe pairs of concepts which cannot appear in conjunction together as they will result in a non locally minimal expression or an unsatisfiable one. The only conjunctive expressions which are minimal and satisfiable are the overlapping pairs, such as $A \sqcap C$. For conjunctions with more than two concepts, each conjunct must overlap with all others for the full expression to be locally minimal, that is for the conjunctive concept $C_1 \sqcap \ldots \sqcap C_n$, all pairs C_i, C_j where $1 \leq i < j \leq n$, we have that $C_i^{\mathcal{J}_\lambda}$ and $C_j^{\mathcal{J}_\lambda}$ overlap. From Figure 4.7, the full set of minimal conjunctions where $n \geq 3$ in this example are:

$$\begin{gathered} A \sqcap C \sqcap \neg D \\ B \sqcap C \sqcap \neg E \\ B \sqcap \neg C \sqcap \neg E \\ C \sqcap \neg D \sqcap \neg E \end{gathered}$$

Therefore, there are 25 possible minimal conjunctive expressions for this example (the single top concept, 8 named atomic and atomic negated concepts, 12 overlapping minimal conjuncts with 2 atoms and 4 overlapping minimal conjunctions of 3 atoms), compared to a total of $2^9 - 1 = 511$ *possible atomic and conjunctive expressions, most of which are unsatisfiable or not locally minimal.*

As shown in Example 4.3.1, the set of relationships determined over the set of concepts applicable to some context λ can be used to inform which concept expressions are minimal with respect to a context-specific interpretation $\mathcal{J}_\lambda$ and local axioms $\mathcal{T}_\lambda$. To tighten the definition of ρ_B so as to restrict the construction of concepts to avoid unwanted refinement steps, we define a new downward refinement operator denoted $\rho_{\bar{\lambda}}(C)$ which is parameterised with the set of most applicable contexts $\bar{\lambda}$ for the subexpression C.

Given a set of most applicable contexts $\bar{\lambda}$ for some subexpression, the set of atomic and negated atomic concepts which were found to be satisfiable in the context referred to by $\bar{\lambda}$ is the intersection of all concepts which were found to be not equivalent to $\perp$ in each set of local axioms $\mathcal{T}_{\lambda'}$ for each $\lambda' \in \bar{\lambda}$. We define the set of such concepts as $arf(\bar{\lambda})$ to represent all *atomic role fillers* as:

$$arf(\bar{\lambda}) \quad = \bigcap\nolimits_{\forall \lambda' \in \bar{\lambda}} \{A \mid A \in sc(\Delta_\lambda) \text{ s.t. } \mathcal{T}_\lambda \not\models A \equiv \perp\}$$

Furthermore, we denote the common set of equivalence, strict subsumption and disjointness axioms ϕ over all local axioms $\mathcal{T}_\lambda$ for all $\lambda' \in \bar{\lambda}$ where $\bar{\lambda} = \{\lambda_1, \ldots, \lambda_n\}$ for

$n \geq 1$ with the shorthand $\mathcal{T}_{\bar{\lambda}}$ as follows:

$$\mathcal{T}_{\bar{\lambda}} = \mathcal{T}_{\lambda_1} \cap \ldots \cap \mathcal{T}_{\lambda_n}$$

By an abuse of notation we also denote the inclusion axioms ϕ entailed by $\mathcal{T}_{\bar{\lambda}}$ with $\cdot_{\mathcal{T}_{\bar{\lambda}}}$ such as $C \equiv_{\mathcal{T}_{\bar{\lambda}}} D$ when $\mathcal{T}_{\bar{\lambda}} \models C \equiv D$.

The operator $\rho_{\bar{\lambda}}$ also makes use of a binary preorder relation $\preceq$ which is imposed on all concepts which can appear as conjuncts or disjuncts. Initially, we introduce how the preorder relation $\preceq$ orders simple concepts in order to describe the first set of cases for $\rho_{\bar{\lambda}}$, but later in Section 4.3.2 will extend it for role expressions, then more complex concept expressions.

Definition 4.3.2. ***(Concept Preorder** $\preceq$**)*** *Given a set of concept expressions S, a **concept preorder** is a binary function $\preceq$: $S \times S$ which imposes an order over concepts, such that for any three concepts $C, D, E \in S$ we have:*

- *Reflexivity: $C \preceq C$*
- *Transitivity: If $C \preceq D$ and $D \preceq E$, then $C \preceq E$*

The concept preorder $\preceq$ can be used by a refinement operator when generating conjunctions (disjunctions) of concepts such that $C_1 \sqcap \ldots \sqcap C_n$ ($C_1 \sqcup \ldots \sqcup C_n$) satisfies $C_i \preceq C_j$ for $1 \leq i < j \leq n$. This ensures, for example, that for concepts A, B where $A \preceq B$, that only $A \sqcap B$ ($A \sqcup B$) is produced where $B \sqcap A$ ($B \sqcup A$) is not, as $A \sqcap B \equiv B \sqcap A$ ($A \sqcup B \equiv B \sqcup A$) under any interpretation we consider, as the logical operators $\sqcap$ and $\sqcup$ are commutative.

We can now describe how the operator $\rho_{\bar{\lambda}}$ is defined for simple concepts in conjunctions as follows:

$$\rho_{\bar{\lambda}}(C) = \begin{cases} \ldots & \\ \{A \mid A \in arf(\bar{\lambda}), A \sqsubset_{\mathcal{T}_{\bar{\lambda}}} C, \neg\exists A' \in arf(\bar{\lambda}) \text{ s.t.} & \text{if } C \in arf(\bar{\lambda}) \\ \quad A \sqsubset_{\mathcal{T}_{\bar{\lambda}}} A' \sqsubset_{\mathcal{T}_{\bar{\lambda}}} C\} \cup \{C \sqcap A \mid C \sqcap A \in arf(\bar{\lambda}), & \\ \qquad C \preceq A, C \text{ overlaps } A\} & \\ \{C_1 \sqcap \ldots \sqcap C_n \sqcap A \mid A \in arf(\bar{\lambda}), & \text{if } C = C_1 \sqcap \ldots \sqcap C_n \\ \quad C_n \preceq A, \forall C_{1 \leq i \leq n} \forall \lambda \in \bar{\lambda} : C_i \in arf(\bar{\lambda}) \rightarrow & \\ \qquad C_i \text{ overlaps } A\} & \\ \ldots & \end{cases}$$

Note that these cases do not yet cover role expressions (§4.3.2), concept disjunction (§4.3.3), or concrete domains (§4.4). In contrast with ρ_B, the operator $\rho_{\bar{\lambda}}$ incorporates context-specific knowledge such as local axioms over the concepts it uses to reduce the chance of generating an improper refinement, or to generate non-minimal or unsatisfiable concepts. In doing so, $\rho_{\bar{\lambda}}$ fails to be *complete* as it explicitly disallows the construction of certain non-minimal or unsatisfiable concepts. This is by design, however, as we wish to prune such concepts from the space under consideration by a refinement operator as they do not aid in the search for solutions for a learning problem.

Example 4.3.3. *Given simple concepts and their relationships from Example 4.3.1 which are ordered as $\top \preceq A \preceq B \preceq C \preceq D \preceq E \preceq \neg C \preceq \neg D \preceq \neg E$, the refinement operator $\rho_{\bar{\lambda}}$ traverses the concept space as follows:*

$$\begin{array}{lllll}
\top & \rightsquigarrow \neg C & \rightsquigarrow \neg C \sqcap \neg E & & \\
 & \rightsquigarrow C & \rightsquigarrow C \sqcap \neg E & & \\
 & & \rightsquigarrow C \sqcap \neg D & \rightsquigarrow C \sqcap \neg D \sqcap \neg E & \\
 & & \rightsquigarrow C \sqcap E & & \\
 & & \rightsquigarrow D & & \\
 & \rightsquigarrow \neg E & \rightsquigarrow A & \rightsquigarrow A \sqcap \neg D & \\
 & & & \rightsquigarrow A \sqcap \neg C & \\
 & & & \rightsquigarrow A \sqcap C & \rightsquigarrow A \sqcap C \sqcap \neg D \\
 & & & \rightsquigarrow D & \\
 & \rightsquigarrow \neg D & \rightsquigarrow \neg D \sqcap \neg E & & \\
 & & \rightsquigarrow B & \rightsquigarrow B \sqcap \neg E & \\
 & & & \rightsquigarrow B \sqcap \neg C & \rightsquigarrow B \sqcap \neg C \sqcap \neg E \\
 & & & \rightsquigarrow B \sqcap C & \rightsquigarrow B \sqcap C \sqcap \neg E \\
 & & & \rightsquigarrow E & \rightsquigarrow E \sqcap \neg C
\end{array}$$

This represents an exhaustive traversal of all 24 unique minimal satisfiable expressions from $\top$.

Despite the use of the preorder $\preceq$ over concepts, Example 4.3.3 demonstrates that the operator $\rho_{\bar{\lambda}}$ generated two refinement chains $\top \rightsquigarrow C \rightsquigarrow D$ and $\top \rightsquigarrow \neg E \rightsquigarrow A \rightsquigarrow D$ which took different paths from $\top$ to D. While this indicates that the refinement operator $\rho_{\bar{\lambda}}$ is *redundant*, it is still designed in such a way as to reduce redundancy, as it was shown in Example 4.3.1 that there are a total of 511 possible expressions instead of the 24 produced by $\rho_{\bar{\lambda}}$ in this case.

4.3.2 Role Expressions

For any set of most applicable contexts $\bar{\lambda}$, the context graph $\mathfrak{G}$ provides a set of role names for use in constructing role expressions of the form $\diamond r.C$ for some quantifier $\diamond$ depending on the chosen concept language. For example, for DLs based on $\mathcal{EL}$, quantifiers may be restricted to $\exists$ only, whereas $\mathcal{ALC}$ also permits $\forall$, and $\mathcal{SROIQ}$ permits both in addition to cardinality quantifiers ${}^{\geqslant n}$, ${}^{\leqslant n}$. A $\mathcal{SROIQ}$ knowledge base may also impose a subsumption hierarchy on role terms, such as $s \sqsubseteq r$ where $r, s \in N_R$. Determining which roles to use together with their quantifiers when refining concepts is largely pre-computed in the structure of a context graph. In this section, we will describe how the information captured in a context graph can be used to extend the definition of the refinement operator $\rho_{\bar{\lambda}}$ for refinements of role expressions.

Recall that in the definition of ρ_B, the set M_B captures atomic concept expressions appropriate for use as starting points for refinement in the context of some role range described by the concept B. The set M_B also includes role expressions $\exists r.(\top)$ and $\forall r.(\top)$ for all roles r with a domain $ad(r)$ not disjoint with B, and are introduced as refinements of the top concept $\top$. While this definition restricts the set of role expressions to those which are appropriate in conjunction with a domain concept B, this is only determined relative to knowledge captured in the TBox. Under a context-specific interpretation we are able to pre-compute tighter restrictions on the set of refinements of role expressions relative to certain contexts and local domains. For example, consider the refinement case for ρ_B which describes refinements of concept expressions C where $C = \exists r.D$:

$$\rho_B(C) = \begin{cases} \dots \\ \{\exists r.E | A = ar(r), E \in \rho_A(D)\} & \text{if } C = \exists r.D \\ \quad \cup \{\exists r.(D) \sqcap E | E \in \rho_B(\top)\} \\ \quad \cup \{\exists s.(D) | s \in sh_{\downarrow}(r)\} \\ \dots \end{cases}$$

Now consider a knowledge base containing role r, concepts B, C, D, E, axioms $C \sqsubseteq B, E \sqsubseteq D \in \mathcal{T}$, where role r has domain B and range D. The operator ρ_B will permit the following two refinement chains:

$$\exists r.D \rightsquigarrow_{\rho_B} \exists r.E$$
$$\exists r.D \rightsquigarrow_{\rho_B} (\exists r.D) \sqcap C$$

Under an open-world interpretation $\mathcal{I}$, the concepts $\exists r.E$ and $(\exists r.D) \sqcap C$ are both satisfiable. However, consider the case where no r-successor is an instance of E, and no instance of C has an r-successor which is an instance of D, as illustrated in Figure 4.8.

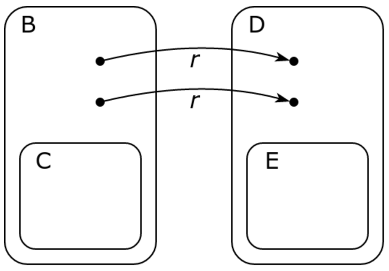


Figure 4.8: Concepts B, C, D, E with their subsumption relationships shown and tuples of role r pairing instances of B and D only.

Under a context-specific interpretation $\mathcal{J}_\lambda$, we would find that $(\exists r.E)^{\mathcal{J}_\lambda} = \emptyset$ and $((\exists r.D) \sqcap C)^{\mathcal{J}_\lambda} = \emptyset$ in this case as no such tuples in $r^{\mathcal{J}_\lambda}$ are known to satisfy these concepts. While this situation can be realised after evaluating such concepts relative to $\mathcal{J}_\lambda$, we do not want the refinement operator to produce them in the first instance if possible, in order to improve the efficiency of a learning algorithm relying on ρ_B to generate such expressions. In this case, we aim to use the structure of the context graph $\mathfrak{G}$ to guide refinements which explicitly captures permissible combinations of concepts and pre-quantified role expressions.

Considering the example illustrated in Figure 4.8, a context graph computed over a knowledge base containing the concepts, role, and role tuples as shown for the role quantifier $\exists$ would contain a vertex for context $\lambda = [\ldots, B]$ and edges (λ, λ') where:

$$\lambda' \in \{[\ldots, B \sqcap \exists r.(\circ), \top], [\ldots, B \sqcap \exists r.(\circ), D], [\ldots, B \sqcap \exists r.(\circ), \neg E]\},$$

In this example, there is no context containing $[\ldots \sqcap \exists r.(\circ), E]$ in the context graph, so the refinement step $\exists r.D \rightsquigarrow \exists r.E$ can be recognised as generating the unsatisfiable subexpression $\exists r.E$. Similarly, as there are no contexts containing $[\ldots, C \sqcap \exists r.(\circ), D]$, the refinement step $\exists r.D \rightsquigarrow (\exists r.D) \sqcap C$ can also be recognised as generating the unsatisfiable subexpression $(\exists r.D) \sqcap C$, which can be avoided.

In general, the set of role expressions $\Diamond r.C$ for some concept C which may be satisfiable in any context λ can be determined directly from the context graph $\mathfrak{G} = (V, E)$ as follows. Given a set of most applicable contexts $\bar{\lambda}$ for any subexpression context, each $\lambda' \in \bar{\lambda}$ may be associated with a set of succeeding contexts $\{\lambda'' \mid (\lambda', \lambda'') \in E\}$. Where any $\lambda' = [D_1, \ldots, D_n]$ for $n \geq 1$, each succeeding context λ'' has the form $\lambda'' = [D_1, \ldots, D_n \sqcap \Diamond r.(\circ_n), D_{n+1}]$. We denote the set of all succeeding role expression fragments of the form $\Diamond r.(D_{n+1})$ given $\bar{\lambda}$ as $re(\bar{\lambda})$ follows:

$$re(\bar{\lambda}) = \bigcap_{\forall \lambda' \in \bar{\lambda}} \{\Diamond r.(D_{n+1}) \mid \forall (\lambda', \lambda'') \in E \text{ where } \lambda'' = [D_1, \ldots, D_n \sqcap \Diamond r.(\circ_n), D_{n+1}]\}$$

As shown in Proposition 4.2.23, if not *all* succeeding contexts λ'' end with $[\ldots, D_n \sqcap \Diamond r.(\circ_n), D_{n+1}]$, then the fragment $\Diamond r.D_{n+1}$ is unsatisfiable in all applicable subexpression contexts $\bar{\lambda}$, and therefore also in the context of the subexpression being refined for which $\bar{\lambda}$ were applicable. Therefore, the set $re(\bar{\lambda})$ contains all role expressions fragments which are at least *permissible*, even though some may be unsatisfiable.

In refinement, new role expressions are introduced either as refinements of the top concept $\top$, or in conjunction with some other expression such as an atomic or negated atomic concept, another role expression, or some conjunction thereof. For downward refinement, we select the most general of all role expressions in $re(\bar{\lambda})$ to introduce into any subexpression context. If the knowledge base supports role inclusions such as $r \sqsupseteq s$ for roles r, s, this information may also be used in selecting the most general set of role expressions. The initial set of role expressions $ir(\bar{\lambda})$ to introduce into downward refinement are the most general of the form $\exists r.D$, $\forall r.D$, and ${\leqslant^n} r.D$, along with any ${\geqslant^n} r.D$ in the absence of any expression $\exists r.D$, as the former is downward refined directly from the latter.

$$ir(\bar{\lambda}) = \{\Diamond r.D \mid \Diamond r.D \in re(\bar{\lambda}) \wedge \neg\exists(\Box s.E) \in re(\bar{\lambda}) \text{ s.t. } (\Diamond r.D) \sqsubseteq_{\mathcal{J}_{\lambda'}} (\Box s.E)\}$$

Role expression subsumption for roles r, s, concepts D, E and any quantifier $\Diamond \in$

$\{\exists, \forall, {\geqslant n}\}$ where $n \geq 1$ can be summarised as follows:

$$\begin{array}{rl}
\Diamond s.D \sqsubseteq \Diamond r.D & \text{when } s \sqsubseteq r \\
\Diamond r.D \sqsubseteq \Diamond r.E & \text{when } D \sqsubseteq E \\
\forall s.D \sqsubseteq \forall r.E & \text{when } s \sqsubseteq r \text{ and } D \sqsubseteq E \\
{\geqslant n} s.D \sqsubseteq {\geqslant m} r.E & \text{when } n \geq m \text{ and } s \sqsubseteq r \text{ and } D \sqsubseteq E \\
{\leqslant n} s.D \sqsupseteq {\leqslant m} r.E & \text{when } n \geq m \text{ and } s \sqsubseteq r \text{ and } D \sqsubseteq E
\end{array}$$

Note that the most general expression of the form ${\leqslant n} r.D$ for $n \geq 0$ is the one where D is as specific as possible, such as when $D = \bot$. Downward refinements of ${\leqslant n} r.D$ require *upward* refinements of D, an operator for which will be presented in Section 4.3.4. To illustrate why this is the case, consider Example 4.3.4.

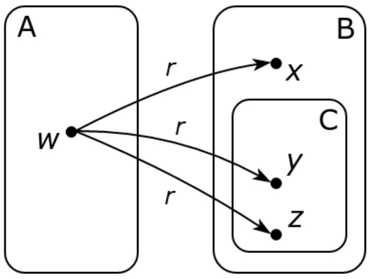


Figure 4.9: An example set of concept names A, B, C with a single role tuple group of three tuples.

Example 4.3.4. *Consider a set of concepts A, B, C where $C \sqsubset B$, together with a single role r and assertions $r(w, x)$, $r(w, y)$, $r(w, z)$, $A(w)$, $B(x)$, $C(y)$, $C(z)$ as illustrated in Figure 4.9. In this example, we have $({\leqslant 2} r.B)^{(\mathcal{I},\mathcal{U})} = \emptyset$ as w has three r-successors in B. Now consider the downward refinement step ${\leqslant 2} r.B \rightsquigarrow_\rho {\leqslant 2} r.C$ where $C \in \rho(B)$ as $C \sqsubset B$. Here, we find that $({\leqslant 2} r.C)^{(\mathcal{I},\mathcal{U})} = \{w\}$ because w has two r-successors in C. Therefore, as $({\leqslant 2} r.C)^{(\mathcal{I},\mathcal{U})} \supseteq ({\leqslant 2} r.B)^{(\mathcal{I},\mathcal{U})}$, the refinement step was generalising (upward). In general, ${\leqslant n} r.C \sqsubseteq {\leqslant n} r.D$ for role r, concepts C, D where $D \sqsubseteq C$, and $n \geq 0$.*

We introduce the most general role expressions of the set $ir(\bar{\lambda})$ in conjunction with simple concepts, other role expressions and conjunctions thereof with new cases for the operator $\rho_{\bar{\lambda}}$, as follows.

$$\rho_{\bar{\lambda}}(C) = \begin{cases} \dots & \\ \{C \sqcap \Diamond r.D \mid \forall(\Diamond r.D) \in ir(\bar{\lambda}) \text{ s.t. } C \preceq (\Diamond r.D)\} & \text{if } C \in arf(\bar{\lambda}) \\ \{C_1 \sqcap \dots \sqcap C_n \sqcap \Diamond r.D \mid \forall(\Diamond r.D) \in ir(\bar{\lambda}) \text{ s.t.} & \text{if } C = C_1 \sqcap \dots \sqcap C_n \\ \quad C_n \preceq (\Diamond r.D)\} & (n \geq 2) \\ \dots & \end{cases}$$

In this way, we rely on the concept precedence operator permitting atomic and negated atomic simple concepts before any role expression, as follows:

$$A \preceq \neg A \preceq \Diamond_1 r.C \preceq \Diamond_2 s.D$$

For any concept name A, concepts C, D, role names $s, r \in N_R$ and quantifiers $\Diamond_1, \Diamond_2$. Note that the precedence operator relies on an ordering of role names $r_1, \dots, r_n$ such that if any $r_i \subseteq r_j$, then $i \leq j$. While the precedence order of differently quantified role expressions is unimportant, we require that they appear after any atomic concept expression A which includes $\top$ or any named concept which is the range of a role expression. In this way, $\preceq$ ensures that multiple role expressions such as $\exists r.D \sqcap \dots \sqcap \exists r.E$ may appear in conjunction, but does not permit both $\exists r.D \sqcap \dots \sqcap \forall s.E$ and $\forall s.E \sqcap \dots \sqcap \exists r.D$ which would lead to redundancy in the search by refinement. Additionally, because of the equivalence $(\forall r.C) \sqcap (\forall r.D) \equiv \forall r.(C \sqcap D)$, the operator will not permit conjunction with an expression of the form $\forall r.C$ if another conjunct $\forall r.D$ is already present.

Once introduced, existing role expressions $\Diamond r.D$ may be downward refined as $\Diamond r.D \rightsquigarrow \Box s.E$ by modifying one of: the quantifier $\Diamond \rightsquigarrow \Box$, the role name $r \rightsquigarrow s$, or the filler $D \rightsquigarrow E$. In each case, the preceding expression will have been associated with a non-empty set of most applicable contexts, $\bar{\lambda}$. For any of these refinements, the set $\bar{\lambda}$ may be different, but must be non-empty as otherwise it indicates that the refinement leads to an unsatisfiable subexpression. For example, consider the refinement of the nested concept expression:

$$C \sqcap \exists r.(D \sqcap \exists s.(E)) \rightsquigarrow C \sqcap {}^{\geqslant 2} r.(D \sqcap \exists s.(E))$$

In this case, all most applicable contexts $\lambda \in \bar{\lambda}$ for the nested subexpression E were those of the form $\lambda = [A_1 \sqcap \exists r.(\circ_1), A_2 \sqcap \exists s.(\circ_2), A_3]$ where $C \sqsubseteq A_1$, $D \sqsubseteq A_2$

and $E \sqsubseteq_{\mathcal{J}_{\bar{\lambda}}} A_3$ in each of their respective contexts. For this such refinement to be permissible, we require that there is a non-empty set of most applicable contexts $\bar{\lambda}'$ for every subexpression. For example, for subexpression E, the non-empty set of most applicable contexts $\bar{\lambda}'$ after this particular refinement must have the form $\lambda' = [A_1 \sqcap {\geqslant}^2 r.(\circ_1), A_2 \sqcap \exists s.(\circ_2), A_3]$. If not, it must have been the case that this particular refinement led to an unsatisfiable subexpression, as no instance chains were found in the instance chase which constructed the context graph which satisfied the nested expression $nest(\lambda')$.

In this way, we are able to use the context graph and most applicable contexts of some subexpression S to determine if any refinement to S leads to any subexpression S' of S which is unsatisfiable. As described in Section 4.2.4, the boolean function $ncu(S)$ evaluates to false if any subexpression S' of S has an empty set of most applicable contexts. In order to compute $ncu(S)$, all most applicable contexts $\bar{\lambda}'$ for every subexpression S' of S need to be recomputed. This is a straightforward procedure which constructs the sets $\bar{\lambda}'$ for each S' by traversing the structure of S and matching most applicable contexts over the edges of the context graph $\mathfrak{G}$ until all subexpressions S' are analysed, which we require for further refinements of the concept containing the subexpressions S' with $\rho_{\bar{\lambda}}$. We now describe the cases for $\rho_{\bar{\lambda}}$

which handle the refinement of role expressions, as follows.

$$\rho_{\bar{\lambda}}(C) = \begin{cases} \ldots & \\ \{C_1 \sqcap \ldots \sqcap \Diamond s.E \sqcap \ldots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \ldots \sqcap \\ \quad \forall(\Diamond s.E) \in \rho_{\bar{\lambda}}(\Box r.D) \text{ s.t.} & \quad \Box r.D \sqcap C_i \sqcap \\ \qquad \forall C_{i \leq j \leq n} \, \Diamond s.E \preceq C_j \wedge & \quad \ldots \sqcap C_n \ (i \leq n) \\ \qquad\quad ncu(C_1 \sqcap \ldots \sqcap \Diamond s.E \sqcap \ldots \sqcap C_n)\} & \\ \{\exists r.E \mid E \in \rho_{\bar{\lambda}'}(D)\} \cup & \text{if } C = \exists r.D \\ \ \{\exists s.(D) \mid s \in sh_{\downarrow}(r)\} \cup & \\ \quad \{{}^{\geqslant n}r.(D) \mid {}^{\geqslant n}r.(D) \in re(\bar{\lambda}) \wedge n \geq 2 \wedge & \\ \qquad \neg\exists({}^{\geqslant m}r.(D)) \in re(\bar{\lambda}) \text{ s.t. } m < n \wedge m \geq 2\} & \\ \{\forall r.E \mid E \in \rho_{\bar{\lambda}'}(D)\} \cup \{\forall s.(D) \mid s \in sh_{\downarrow}(r)\} & \text{if } C = \forall r.D \\ \{{}^{\geqslant n}r.D \mid {}^{\geqslant n}r.D \in re(\bar{\lambda}) \wedge n > m \wedge & \text{if } C = {}^{\geqslant m}r.D \\ \qquad \neg\exists({}^{\geqslant o}r.(D)) \in re(\bar{\lambda}) \text{ s.t. } m < o < n\} \cup & \\ \ \{{}^{\geqslant m}s.D | s \in sh_{\downarrow}(r)\} \cup \{{}^{\geqslant m}r.E \mid E \in \rho_{\bar{\lambda}'}(D)\} & \\ \{{}^{\leqslant n}r.D \mid {}^{\leqslant n}r.D \in re(\bar{\lambda}) \wedge n < m \wedge & \text{if } C = {}^{\leqslant m}r.D \\ \qquad \neg\exists({}^{\leqslant o}r.(D)) \in re(\bar{\lambda}) \text{ s.t. } n < o < m\} \cup & \\ \ \{{}^{\leqslant m}s.D | s \in sh_{\downarrow}(r)\} \cup \{{}^{\leqslant m}r.E \mid E \in \upsilon_{\bar{\lambda}'}(D)\} & \\ \ldots & \end{cases}$$

In the definition above, the set of most applicable contexts $\bar{\lambda}'$ denotes those for the interior expression $\circ$ being refined in the role expression $\Diamond r.(\circ)$. Note also that downward refinements of maximum qualified cardinality role expression ${}^{\leqslant n}r.C$ relies on the use of the upward refinement operator $\upsilon_{\bar{\lambda}}$, which is described in Section 4.3.4.

With most of the cases for $\rho_{\bar{\lambda}}$ now defined, we now turn our attention to disjunctive subexpressions of the form $C_1 \sqcup \ldots \sqcup C_n$ for $n \geq 2$ in Section 4.3.3.

4.3.3 Disjunctive Expressions

To extend the operator $\rho_{\bar{\lambda}}$ to handle disjunctive expressions, we follow the definition of ρ_B which only permits the construction of disjunctions as refinements of $\top$, as follows.

$$\rho_B(C) = \begin{cases} \dots \\ \{C_1 \sqcup \dots \sqcup C_n \mid C_i \in M_B (1 \leq i \leq n\} & \text{if } C = \top \\ \{C_1 \sqcup \dots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \dots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \dots \sqcup C_n \ (n \geq 2) \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} \cup & \\ \quad \{(C_1 \sqcup \dots \sqcup C_n) \sqcap D | D \in \rho_B(\top)\} & \\ \dots \end{cases}$$

We observe that refinements of disjunctions with the conjunction of another concept expression such as $(C_1 \sqcup \dots \sqcup C_n) \rightsquigarrow (C_1 \sqcup \dots \sqcup C_n) \sqcap D$ is simply a short-cut to refining in n steps to produce $((C_1 \sqcap D) \sqcup \dots \sqcup (C_n \sqcap D))$ because conjunction is distributive over disjunction where $(C \sqcup D) \sqcap E \equiv (C \sqcap E) \sqcap (D \sqcap E)$, therefore we omit this case.

In the case of $\rho_{\bar{\lambda}}$, we permit the introduction of disjunctive expressions from the set of any most general atomic concepts permissible in any context λ, which we define as $mga(\bar{\lambda})$ where:

$$mga(\bar{\lambda}) = \{A \mid \forall A \in arf(\bar{\lambda}), \neg\exists A' \in arf(\bar{\lambda}) \text{ s.t. } A \sqsubset_{\mathcal{J}_{\bar{\lambda}}} A'\}$$

Note that the set $mga(\bar{\lambda})$ may contain $\top$ as it is always the most general concept. We now define the handling of disjunctions with $\rho_{\bar{\lambda}}$ as follows.

$$\rho_{\bar{\lambda}}(C) = \begin{cases} \dots \\ \{C_1 \sqcup \dots \sqcup C_n \mid C_{1 \leq i \leq n} \in mga(\bar{\lambda}) & \text{if } C = \top \\ \qquad \forall C_{1 \leq i < j \leq n} : C_i \preceq C_j\} & \\ \{C_1 \sqcup \dots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \dots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \dots \sqcup C_n \\ \qquad D \in \rho_{\bar{\lambda}}(C_i) \wedge C_{i-1} \preceq D \preceq C_{i+1}, 1 \leq i \leq n\} & \quad (n \geq 2) \\ \dots \end{cases}$$

Note that these rules permit the introduction of disjunctions such as $A \sqcup A \sqcup \dots \sqcup A$ for the same atom A. Clearly, such a refinement step is improper, but is necessary in order to reach concepts or subexpressions such as $A_1 \sqcup \dots \sqcup A_n$ where each A_i is some unique complex expression such as $A \sqcap \Diamond r.(B)$. In a search procedure, introducing disjunctive expressions with a large number of disjuncts n significantly expands the search space. Therefore, we may permit a user to manually impose a global limit on n, the number of disjuncts in any disjunction, to limit the search space.

Furthermore, improper refinements are possible, for example, when refining $A \sqcup B \rightsquigarrow A \sqcup D$ where $D \in \rho_{\bar{\lambda}}(B)$ and $D \sqsubseteq B \sqsubseteq A$. In this case, $(A \sqcup B)^{\mathcal{J}_\lambda} = (A \sqcup D)^{\mathcal{J}_\lambda}$ in any context λ as A is the subsuming concept. Any number of refinements of the disjunct B will always produce an equivalent expression, and therefore represents improper refinement steps.

When applying such refinement operators to a learning problem we may wish to avoid such refinements as they are not productive, in that they do not alter the coverage of the concept containing the refined subexpression $A \sqcup D$ yet they increase the complexity of the subexpression in an unguided way. We tackle this problem with a method of altering the behaviour of $\rho_{\bar{\lambda}}$ by *suspending* refinement of any disjunctive operand which is strictly subsumed by another. However, testing the subsumption of concepts without first computing their interpretation under $\mathcal{J}_\lambda$ is difficult, as it relies on a description logic reasoner to compute subsumption under an open-world interpretation which can be computationally expensive. An alternative may be to use *structural subsumption* to test if some operand B subsumes D based on its structure alone, however such algorithms are difficult to define, and as recognised in work which defines ρ_B [58], no tractable complete structural subsumption algorithms exist for the DL $\mathcal{ALC}$ [6]. As we are considering more expressive concept languages than $\mathcal{ALC}$, this is not a feasible solution either. In Chapter 5, Section 5.3.2, we introduce a novel method which permits us to recognise when operands of disjunctions are subsumed in any context by persisting information captured during the computation of their coverage, which is used in subsequent refinements.

4.3.4 Upward Refinement

In the definition of the downward refinement operator $\rho_{\bar{\lambda}}$, we encountered a need to refine *upwards* such as when downward refining maximum qualified cardinality restrictions $\leqslant^n r.D$ as shown in Example 4.3.4). In this section, we will define an upward operator for use in these circumstances to complement the overall construction of $\rho_{\bar{\lambda}}$.

Upward refinement, or concept generalisation with operator $\upsilon_{\bar{\lambda}}$ is not as straightforward to define as downward refinement as we have presented with the definition of $\rho_{\bar{\lambda}}$. Typically, upward refinement begins with refinements of the bottom concept $\perp$, just as downward refinement begins with refinements of most general concepts like $\top$. In order to generalise $\bot \rightsquigarrow_{\upsilon_{\bar{\lambda}}} C$ in a step-wise manner for some C, we require that each concept C be as specific as possible at each step.

The problem of generalisation has been tacked in ILP before. Amongst the techniques considered are ones which generalise by first computing highly specific concepts which tightly describe example instances in the domain, which in our case corresponds to each individual i where $i \in \Delta_\lambda$. Over these specific concepts, a generalisation operator is then applied to produce concepts which subsume subsets of specific concepts progressively until either solutions are found, or the most general concept is reached [67]. However, when $|\Delta_\lambda|$ is very large for some context λ, and the concept language has high expressivity such as those we are considering, this approach will likely result in the generation of a vast set of concepts for upward refinement which may be inefficient to traverse.

In order to avoid the potential inefficiency of generalisation over the large space of possible concepts in any context λ, we intentionally define our upward refinement operator $\upsilon_{\bar{\lambda}}$ to be *incomplete* by generalising from specific concepts which are composed exclusively of simple concepts from $arf(\bar{\lambda})$ without role expressions. The rationale behind this approach is that we may still want to take advantage of maximum qualified cardinality restrictions ${\leqslant^n} r.C$ in our hypothesis language, but also wish to restrict the range of concepts that C may take on to limit the overall search space of concepts, as has been a goal throughout this chapter. We leave the definition of a complete upward refinement operator for DL concepts to future work.

We begin by defining the set of most specific concept expressions in a set of most applicable contexts $\bar{\lambda}$ composed exclusively of concepts from $arf(\bar{\lambda})$ as the set $msc(\bar{\lambda})$ as follows:

$$msc(\bar{\lambda}) \quad = \{A_1 \sqcap \ldots \sqcap A_n \mid \forall i \in \Delta_{\bar{\lambda}}\ \forall_{1 \leq j < k \leq n}\ A_j \in atom(i) \wedge \neg \exists A_k \text{ s.t. } A_j \sqsubset_{\mathcal{J}_{\bar{\lambda}}} A_k\}$$

The set $msc(\bar{\lambda})$ provides us with all the most specific locally minimal conjunctions of simple concepts which are permissible in the context described by all the most applicable contexts $\bar{\lambda}$.

Example 4.3.5. *Given the set of concepts and strict subsumption, overlap and disjointness relationships from Example 4.3.1, the set of most specific concepts are:*

$$\begin{aligned} msc(\bar{\lambda}) \quad &= \{D, (C \sqcap E), (A \sqcap \neg C), (E \sqcap \neg C), \\ &\quad (A \sqcap C \sqcap \neg D), (C \sqcap \neg D \sqcap \neg E), (B \sqcap \neg C \sqcap \neg E), (B \sqcap C \sqcap \neg E)\} \end{aligned}$$

This set can be compared to the top-down traversal of the downward operator $\rho_{\bar{\lambda}}$ of Example 4.3.3 where each of the concepts in $msc(\bar{\lambda})$ appear at the end of downward refinement chains.

The upward refinement operator $\upsilon_{\bar{\lambda}}$ employs the set $msc(\bar{\lambda})$ as starting points in refinement from $\bot$ to conjunctions in the same way that $\rho_{\bar{\lambda}}$ uses the set $mgc(\bar{\lambda})$ in generating refinements of $\top$ to disjunctions. From these concepts, generalisation operators either remove a conjunct, replace a single atomic or negated atomic concept with a subsuming concept from $arf(\bar{\lambda})$, or create a disjunction with an element from $msc(\bar{\lambda})$ if it is not already subsumed as shown in the definition of $\upsilon_{\bar{\lambda}}$ below.

Definition 4.3.6. *Given a concept expression C and a set of most applicable contexts $\bar{\lambda}$, the upward refinement operator $\upsilon_{\bar{\lambda}}(C)$ is defined as:*

$$\upsilon_{\bar{\lambda}}(C) = \begin{cases} \{D \mid D \in msc(\bar{\lambda})\} & \textit{if } C = \bot \\ \{A' \mid A' \in sh_{\uparrow}(A)\} \cup & \textit{if } C = A, A \in msc(\bar{\lambda}) \\ \quad \{C \sqcup A' \mid A' \in msc(\bar{\lambda}), A' \not\sqsubseteq_{\mathcal{J}_{\bar{\lambda}}} C\} & \\ \{C_1 \sqcap \ldots \sqcap C_{i-1} \sqcap C_{i+1} \sqcap C_n \mid 1 \leq i \leq n\} \cup & \textit{if } C = C_1 \sqcap \ldots \sqcap C_n (n \geq 2) \\ \quad \{C \sqcup A' \mid A' \in msc(\bar{\lambda}), A' \not\sqsubseteq_{\mathcal{J}_{\bar{\lambda}}} C\} & \\ \{C_1 \sqcup \ldots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup C_n | D \in \upsilon_{\bar{\lambda}}(C_i), & \textit{if } C = C_1 \sqcup \ldots \sqcup C_n (n \geq 2) \\ \quad 1 \leq i \leq n, \forall C_{1 \leq i \leq n} : D \not\sqsubseteq_{\mathcal{J}_{\lambda}} C_i\} \cup & \\ \qquad \{C_1 \sqcup \ldots \sqcup C_n \sqcup A | A \in msc(\bar{\lambda}), & \\ \qquad\quad \forall C_{1 \leq i \leq n} : A \not\sqsubseteq_{\mathcal{J}_{\lambda}} C_i\} & \end{cases}$$

The definition of operator $\upsilon_{\bar{\lambda}}$ is designed to reduce potential occurrences of improper refinements when refining or generating disjuncts, as it ensures that disjuncts are not subsumed by some other disjunct. This ensures that the operator does not generate expressions such as $C \sqcup D$ where $D \sqsubseteq_{\mathcal{J}_{\bar{\lambda}}} C$, as otherwise $(C \sqcup D) \equiv C$.

4.4 Learning Over Concrete Domains

One aspect of concept refinement we have not yet addressed is the handling of expressions over concrete domains such as number, boolean or string literals. We aim to improve methods for concept search by refinement over knowledge bases with potentially large data sets capturing experimental data which is likely to make use of concrete domains, so we will incorporate them into the definition of the $\rho_{\bar{\lambda}}$ refinement operator. The DL-Learner system employs methods of handling such domains as part of the downward refinement operator ρ_B [56], which we will now describe.

Ordered sets of literals such as the integers $\mathbb{Z}$, reals $\mathbb{R}$, boolean values $\mathbb{B}$ and strings $\mathbb{S}$ are all examples of *concrete domains* in OWL for which so-called *datatype* properties range over. Similarly, in a DL, datatype roles are those which pair abstract individuals from Δ to literals from some concrete domain, as described in Definition 3.2.6.

At the time of writing, the DL-Learner system currently supports real (double) and boolean literals accessible as d-successors of any datatype role d. Real-valued literals are analysed as $values_d$ where $values_d = \{l | \mathcal{K} \models d(i,l)\}$ for real-valued l sorted in ascending order by $<$. When viewed as a list, the ordered set $values_d$ can be indexed to refer to the i^{th} element with $values_d[i]$. In order to generate *facet restrictions* (§3.2.1) over this list for use in refinement, a pre-determined number of splits s_{max} is specified and used to limit the number of ways a refinement operator can partition $values_d$ with facet restrictions such as $\leq v$ and $\geq v$ into a list of pre-selected values for datatype role d as $splits_d$, which is defined as follows:

$$splits_d = \{t_j | i = \frac{|values_d|}{s_{max}+1}, t_j = 0.5 \cdot (values_d[\lfloor i \cdot j \rfloor] + values_d[\lfloor i \cdot j \rfloor + 1]), 1 \leq j \leq s_{max}\}$$

where $\lfloor v \rfloor$ is the *floor* function which rounds real values down to the closest integer. The downward refinement operator ρ_B then incorporates the list of pre-calculated list of split values $splits_d$ for the datatype roles d against facet restrictions such as $\leq v$, $\geq v$ where $v \in splits_d$, along with boolean value facet restrictions $(= true)$, $(= false)$ as follows:

$$\rho_B(C) = \begin{cases} \dots & \\ \emptyset & \text{if } C = (= true) \\ \emptyset & \text{if } C = (= false) \\ \{\geq w \mid v = splits_d[i], i > 1, w = splits_d[i-1]\} & \text{if } C = (\geq v) \\ \{\leq w \mid v = splits_d[i], i < \sharp splits_d, w = splits_d[i+1]\} & \text{if } C = (\leq v) \\ \dots & \end{cases}$$

The set of terms used as initial expressions for refinement in M_B is also extended to contain the following facet restrictions, depending on the datatype of range expression B for datatype role d which is assumed to be either only boolean or double:

$$
\begin{array}{ll}
\{(= \textit{true}), (= \textit{false})\} & \text{(if } d \text{ ranges over boolean values)} \\
\{\geq \textit{splits}_d[s_{max}], \leq \textit{splits}_d[1]\} & \text{(if } d \text{ ranges over double values)}
\end{array}
$$

As recognised in the presentation of this particular method for handling numerical facet restrictions by refinement, the pre-determined *splits* method was chosen to limit the number of steps a refinement operator takes if there are very many literals in the domain $\textit{values}_d$ for some datatype role d. However, depending on the number of splits chosen, this approach may permit sub-optimal choices of numerical facet restriction values if optimal choices lie between two adjacent split values $\textit{splits}_d[i]$ and $\textit{splits}_d[i+1]$.

In our work, we aim to significantly improve upon the handling of numerical facet restrictions by the refinement operator $\rho_{\bar{\lambda}}$ by taking advantage of information in the context graph and associated data structures which capture a more detailed distribution of literals in each subexpression context λ. We begin by describing an extension to the closed-world context-specific interpretation $\mathcal{J}_\lambda$ for handling facet restrictions over boolean, string, integer and double concrete domains in each context λ.

4.4.1 Context-Specific Interpretation of Datatype Expressions

Datatype restrictions in OWL2-DL and the underlying logic $\mathcal{SROIQ}(D)$ permit expressions which combine facet restrictions appropriate for particular concrete domains $\mathcal{D}$ with boolean connectives such as conjunction $\wedge$ and disjunction $\vee$. For example, the domain of boolean values $\mathbb{B} = \{\textit{true}, \textit{false}\}$ permit the facet restriction $=$, the domain of finite strings $\mathbb{S}$ permits $=$ along with regular expressions to match particular strings $s \in \mathbb{S}$, and numerical domains such as the integers $\mathbb{Z}$ and reals $\mathbb{R}$ permit the facet restrictions $\leq, <, >, \geq$. Restrictions over a common concrete domain can then be combined with connectives to represent new expressions such as $\textit{double}[(\geq 4.2 \wedge < 5.1) \vee (> 5.2))]$ which represents the two sets of double-precision real-typed literals $[4.2, 5.1)$ and $(5.2, \infty)$. The closed-world context-specific interpretation of such expressions is as follows, where R, S are facet restrictions over a common concrete domain $\mathcal{D} \in \{\mathbb{B}, \mathbb{S}, \mathbb{Z}, \mathbb{R}\}$:

$$\begin{array}{ll}
(= \mathit{true})^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{B}} \mid v = \mathit{true}\} & (= \mathit{false})^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{B}} \mid v = \mathit{false}\} \\
(= s)^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{S}} \mid v = s\} & \\
(\mathit{int}[> n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{Z}} \mid v > n\} & (\mathit{double}[> n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{R}} \mid v > n\} \\
(\mathit{int}[< n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{Z}} \mid v < n\} & (\mathit{double}[< n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{R}} \mid v < n\} \\
(\mathit{int}[\geq n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{Z}} \mid v \geq n\} & (\mathit{double}[\geq n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{R}} \mid v \geq n\} \\
(\mathit{int}[\leq n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{Z}} \mid v \leq n\} & (\mathit{double}[\leq n])^{\mathcal{J}_\lambda} = \{v \in \Delta_\lambda^{\mathbb{R}} \mid v \leq n\} \\
(R \wedge S)^{\mathcal{J}_\lambda} = R^{\mathcal{J}_\lambda} \cap S^{\mathcal{J}_\lambda} & (R \vee S)^{\mathcal{J}_\lambda} = R^{\mathcal{J}_\lambda} \cup S^{\mathcal{J}_\lambda}
\end{array}$$

Note that we do not interpret regular expressions over strings in our interpretation, only equality, as we do not consider refinements of regular expressions in our learning algorithm. It is plausible, however, that such a refinement may be useful for a learner over the domain of natural or computer languages.

OWL2 also supports date and time literals which can be constrained by facet restrictions similar to those with numbers. Again, we do not consider them here, but they could be included in a straightforward way [66]. Perhaps with the exception of dates and times, the interpretation $\mathcal{J}_\lambda$ over boolean, string, integer and double domain restrictions provides us with a degree of expressivity suitable for posing simple restrictions over most common data types we expect to encounter in knowledge bases.

Definition 4.2.8 which describes the interpretation $\mathcal{J}_\lambda$ over concept terms extends naturally with the concrete domain interpretations above, permitting concepts such as:

Concept	*Interpretation*
$(\exists r.(\mathit{boolean}[= \mathit{true}]))^{\mathcal{J}_\lambda}$	All individuals i in Δ_λ which have at least one r-successor literal equal to *true*.
$(\forall r.(\mathit{int}[> 5]))^{\mathcal{J}_\lambda}$	All individuals i in Δ_λ for which all r-successors are integer literals greater than 5.
$({}^{\leq 2}r.(\mathit{double}[\leq -2.2]))^{\mathcal{J}_\lambda}$	All individuals i in Δ_λ which have at most two r-successors which are double values less than or equal to -2.2.

Note that the set of literals applicable in any subexpression context λ ending with a datatype role d is Δ_λ which differs from the sets values_d for datatype roles d in the definition of ρ_B, as it is limited to only those literals which are applicable, or reachable from the set of examples, in each context λ. This allows us to partition the literal data in the interpretation $(\mathcal{I},\mathcal{U})$ for any knowledge base into specific subsets Δ_λ which permits us to define context-dependent splitting methods over relevant subsets of literals.

The instance chase of Algorithm 4 describes how to construct local domains Δ_λ for every context λ captured in a context graph $\mathfrak{G}$. The construction of each local domain Δ_λ over datatype literals affords the benefit of understanding the limited set of literals in each context over which to describe facet restrictions.

Given an exact subexpression context λ which corresponds to a role filler as a quantified datatype literal such as $\lambda = [D_1, \ldots, D_n \sqcap \Diamond d.(\circ_n), \top^{\mathcal{D}}]$ for some datatype $\mathcal{D}$, we know from Corollary 4.2.21 that an approximation of the local domain Δ_λ is the intersection of all local domains for all most applicable contexts $\bar{\lambda}$ describing λ from the context graph. Therefore, we can compute the limited set of literals $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ by taking the intersection of all $\Delta^{\mathcal{D}}_{\lambda'}$ for all $\lambda' \in \bar{\lambda}$.

Once computed, the limited set of literals in each $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ can be used to pre-determine appropriate splits for refinement. However, we aim to take the computation of facet restrictions in each context a step further by recognising that Algorithm 4 attributes to each literal in each local domain $\Delta^{\mathcal{D}}_{\lambda'}$ the set of example individuals $e \in \mathcal{E}$ from which they were reachable. This information is useful given the intended use of the concepts being refined, which is to act as hypotheses in a classification or subgroup discovery learning problem as we will present in Chapter 5. In these learning problems, the sets of examples are labelled with symbols from Ω and concepts are often sought which describe the individuals and literals reachable from examples with one label $\omega \in \Omega$ to the exclusion of those from all others $\Omega \setminus \{\omega\}$. Given any set of numerical literals $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ where $D \in \mathbb{Z}, \mathbb{R}$ ordered by $\leq$, we can identify *runs* of literals which are reachable only from examples of one label l, as shown in Example 4.4.1.

Example 4.4.1. *Consider a set of doubles $\Delta^{\mathbb{R}}_{\bar{\lambda}} = \{-0.2, 1.3, 3.5, 4.0, 7.7, 10.6, 11.2\}$, a set of examples labels $\Omega = \{+, -\}$ and the set of examples $\mathcal{E}$ which are partitioned into two labelled sets as $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, and where an instance chase procedure has attributed to each of the doubles the following labels as being reachable from a labelled example $e \in \mathcal{E}^+$ or $e \in \mathcal{E}^-$ any of the contexts $\lambda' \in \bar{\lambda}$ as follows:*

Literal:	-0.2	1.3	3.5	4.0	7.7	10.6	11.2
Labels:	$+$	$+,-$	$+,-$	$+$	$+$	$-$	$-$

All of the doubles reachable from examples labelled with either $+$ or $-$ can be described with *the following facet restrictions:*

Label	*Restriction*
$+$	$double[< 10.6]$
$-$	$double[> -0.2]$

A downward refinement of either of these restrictions targeting a particular label $+$ or $-$ may be the following:

Label	*Refinement step*	*Excluded*
$+$	$double[< 10.6] \rightsquigarrow double[(< 1.3) \vee (> 1.3 \wedge < 10.6)]$	$\{1.3\}$
$+$	$double[< 10.6] \rightsquigarrow double[(< 3.5) \vee (> 3.5 \wedge < 10.6)]$	$\{3.5\}$
$-$	$double[> -0.2] \rightsquigarrow double[(> -0.2 \wedge < 4.0) \vee (> 7.7)]$	$\{4.0, 7.7\}$

In each of these cases, the refinement steps excluded a ***single largest contiguous run of literals labelled with the opposite label*** *from the set covered by the preceding expression.*

Example 4.4.1 demonstrated that numerical facet restrictions can be constructed from any approximating local domain $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ based on the example labels from Ω attributed to each literal in the set. We denote the technique for constructing refinements of numerical domains in this way the *run exclusion method*. To describe this method, we first define several functions which aid in its explanation.

For simplicity, we denote the label ω where $\omega \in \Omega$ of any example $e \in \mathcal{E}$ as being computable by function $label(e)$ defined as follows:

$$label(e) = \omega \text{ where } e \in \mathcal{E}^{\omega}$$

The function $label(e)$ is deterministic for any example e as we assume all examples are only given one label each, and the sets $\mathcal{E}^{\omega}$ for all $\omega \in \Omega$ are pairwise disjoint. However, any single literal v may be reachable via instance chains from several examples, therefore may be attributed with multiple labels from Ω by the instance chase. The set of labels attributed to any single literal v amongst a set of most applicable contexts $\bar{\lambda}$ is retrievable by the function $labels(v, \bar{\lambda})$ defined as follows:

$$labels(v, \bar{\lambda}) = \{\omega \mid \forall \lambda \in \bar{\lambda}, \exists (\lambda, v, e) \in L \text{ where } label(e) = \omega\}$$

where the set L contains triples (λ, v, e) which denote that individual or literal v in the local domain Δ_{λ} for subexpression context λ was reachable from example e, and was constructed by the instance chase of Algorithm 4. The set of literals labelled with ω common to all contexts $\lambda \in \bar{\lambda}$ is denoted $literals(\omega, \bar{\lambda})$ as follows:

$$literals(\omega, \bar{\lambda}) = \{v \mid \forall v \in \Delta^{\mathcal{D}}_{\bar{\lambda}} \text{ where } \omega \in labels(v, \bar{\lambda})\}$$

Assuming a non-empty numerical domain $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ and non-empty subset $literals(\omega, \bar{\lambda})$, we denote the global minimum g_{min} and maximum g_{max}, and labelled minimum l^{ω}_{min} and labelled maximum l^{ω}_{max} as follows:

$$\begin{aligned} g_{min} &= min\ \Delta^{\mathcal{D}}_{\bar{\lambda}} & \qquad l^{\omega}_{min} &= min\ literals(\omega, \bar{\lambda}) \\ g_{max} &= max\ \Delta^{\mathcal{D}}_{\bar{\lambda}} & \qquad l^{\omega}_{max} &= max\ literals(\omega, \bar{\lambda}) \end{aligned}$$

For some example label ω we may compute the lower bound $lb(\omega, \bar{\lambda})$ and upper bound $ub(\omega, \bar{\lambda})$ over the domain $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ as follows:

$$lb(\omega, \bar{\lambda}) = \begin{cases} -\infty & \text{if } l^{\omega}_{min} = g_{min} \\ l^{\omega}_{min} & \text{otherwise} \end{cases} \qquad ub(\omega, \bar{\lambda}) = \begin{cases} \infty & \text{if } l^{\omega}_{max} = g_{max} \\ l^{\omega}_{max} & \text{otherwise} \end{cases}$$

We denote the set of predecessor and successor literals $pre(v, \bar{\lambda})$ and $suc(v, \bar{\lambda})$ over an entire domain $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ as follows:

$$\begin{aligned} pre(v, \bar{\lambda}) &= \{v' \mid v' \in \Delta^{\mathcal{D}}_{\bar{\lambda}} \text{ where } v' < v\} \\ suc(v, \bar{\lambda}) &= \{v' \mid v' \in \Delta^{\mathcal{D}}_{\bar{\lambda}} \text{ where } v' > v\} \end{aligned}$$

Given any label $\omega \in \Omega$, we can now define the lower bound $lb(\omega, \bar{\lambda}) = v^{\omega}_{min}$ and upper bound $ub(\omega, \bar{\lambda}) = v^{\omega}_{max}$ for all literals with label ω over set $\Delta^{\mathcal{D}}_{\bar{\lambda}}$. Then, again for any individual label ω, we can compute the set of *most general numerical facet restrictions* with function $mgnfr(\omega, \bar{\lambda})$ as follows where t is a numerical datatype, either $t = int$ when $\mathcal{D} = \mathbb{Z}$ or $t = double$ when $\mathcal{D} = \mathbb{R}$:

$$mgnfr(\omega, \bar{\lambda}) = \begin{cases} \top^{\mathcal{D}} & \text{if } v^{\omega}_{min} = -\infty \wedge v^{\omega}_{max} = \infty \\ t[> max\ pre(v^{\omega}_{min}, \bar{\lambda})] & \text{if } v^{\omega}_{max} = \infty \wedge pre(v^{\omega}_{min}, \bar{\lambda}) \neq \emptyset \\ t[< min\ suc(v^{\omega}_{max}, \bar{\lambda})] & \text{if } v^{\omega}_{min} = -\infty \wedge suc(v^{\omega}_{max}, \bar{\lambda}) \neq \emptyset \\ t[\geq v^{\omega}_{min} \wedge \leq v^{\omega}_{max})] & \text{if } pre(v^{\omega}_{min}, \bar{\lambda}) = suc(v^{\omega}_{max}, \bar{\lambda}) = \emptyset \\ t[> max\ pre(v^{\omega}_{min}, \bar{\lambda}) \wedge & \text{otherwise} \\ \quad < min\ suc(v^{\omega}_{max}, \bar{\lambda})] & \end{cases}$$

Most general numerical facet restrictions describe starting points in refinement which cover all literals in a context for each label $\omega \in \Omega$. As demonstrated in Example 4.4.1, subsequent refinements of most general numerical facet restrictions generated to cover literals with label ω will seek to exclude the largest contiguous run of literals with *other* labels $\omega' \in \Omega$ where $\omega' \neq \omega$. This strategy aligns with the motivation for classification and subgroup discovery learning problems where we aim to generate concepts which describe the literals reachable from examples of one label to the exclusion of any others.

The kinds of numerical facet restrictions we consider are only those which are disjunctions of expressions $N = t[F_1 \vee \ldots \vee F_n]$ for $n \geq 1$ initially constructed by $mgnfr(\omega, \bar{\lambda})$ to target literals with label ω. Once this is performed, the expression N and all of its subexpressions and refinements thereof will be attributed with label ω, denoting the *target label* of the intended labelled literals in the interpretation of N, which we denote with the subscript N_ω.

Any F_i in a targeted numerical facet restriction expression $N_\omega = t[F_1 \vee \ldots \vee F_n]$ for $1 \leq i \leq n$ must represent a range of numbers which is either bounded above, below, or both as a conjunction of facet restrictions. Specifically, each F_i must take the form $F_i = f_{min}$, $F_i = f_{max}$, or $F_i = (f_{min} \wedge f_{max})$ where $f_{min} \in \{> v_{min}, \geq v_{min}\}$ and $f_{max} \in \{< v_{max}, \leq v_{max}\}$ which, when appearing together in a disjunction, describe non-overlapping numerical ranges with an inclusive or exclusive lower bound v_{min}, an inclusive or exclusive upper bound v_{max}, or a combination of the two. Each numerical range expression F_i must cover a non-empty set of literals from the approximate local domain where $N_\omega^{\mathcal{I}_{\bar{\lambda}}} \subseteq \Delta_{\bar{\lambda}}^{\mathcal{D}}$ for the set of most applicable contexts $\bar{\lambda}$ as retrieved from a context graph.

In order to downward refine any N_ω, we modify one of the interior conjunctive expressions F_i and compute the set of longest contiguous runs of literals labelled with any *other* label ω' where $\omega' \in \Omega$ and $\omega' \neq \omega$. We define *contiguous* to mean any unbroken sequence of literals $v_i, \ldots, v_j$ of length $(j - i) \geq 1$ from a set of literals ordered by $<$ such as $V = v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n$ for $1 \leq i \leq j \leq n$ where each literal v_i to v_j shares a common label ω' and which are bounded by v_{i-1} and v_{j+1} labelled with ω. In Example 4.4.1, the ordered sets $\{1.3\}$ and $\{3.5\}$ were examples of largest contiguous runs of literals labelled with $-$ amongst those labelled with $+$, and the ordered set $\{4.0, 7.7\}$ was the only largest contiguous run of literals labelled with $+$ amongst those labelled with $-$. Given an ordered set of labelled numerical literals V, the function $contig(\omega, V)$ can be used to determine *all* contiguous runs of literals labelled with ω and is described in Algorithm 5.

Algorithm 5 The $contig(\omega, V)$ function to compute the set of all contiguous non-empty runs of numerical literals labelled with ω from a non-empty ordered sequence of numerical literals $V = \{v_1, \ldots, v_n\}$ where $n \geq 1$.

```
function $contig(\omega, \{v_1, \ldots, v_n\})$
    $S := \emptyset$                                        ▷ The set of contiguous runs in $V$
    $R := \emptyset$                                        ▷ The current set of literals in a run
    $i := 1$                                                ▷ Literal index starting with $v_1$
    while $i \leq n$ do
        if $\omega \in labels(v_i) \wedge \exists \omega' \in labels(v_i)$ s.t. $\omega' \neq \omega$ then
            if $R \neq \emptyset$ then
                $S := S \cup \{R\}$
                $R := \emptyset$
            end if
            $S := S \cup \{\{v_i\}\}$
        else if $\omega \in labels(v_i)$ then
            $R := R \cup \{v_i\}$
        else if $R \neq \emptyset$ then
            $S := S \cup \{R\}$
            $R := \emptyset$
        end if
        $i := i + 1$
    end while
    if $R \neq \emptyset$ then
        $S := S \cup \{R\}$
    end if
    return $S$
end function
```

The sets of all equally longest contiguous runs of a set $contig(\omega, V)$ is defined as $contig_{max}(\omega, V)$ as follows:

$$contig_{max}(\omega, V) = \{R \mid \forall R \in contig(\omega, V) \text{ where } \neg\exists R' \text{ s.t. } |R'| > |R|\}$$

Downward refinements of a disjunctive numerical facet expression $N_\omega = t[F_1 \vee \ldots \vee F_n]$ can be performed for each F_i where $1 \leq i \leq n$ by the function $split(F_i, \omega, \Omega, \bar{\lambda})$ as described by Algorithm 6. For each F_i which is intended to target literals labelled only with ω, the *split* algorithm targets all other literals labelled with $\omega' \in \Omega \setminus \{\omega\}$ by constructing sets of longest contiguous runs of literals with $contig_{max}(\omega', F_i^{\mathcal{I}_{\bar{\lambda}}})$, then subtracts these from $F_i^{\mathcal{I}_{\bar{\lambda}}}$ to produce new literal sets to be covered by the refined expression F_i'. The *split* algorithm makes use of the function *rnr* which performs the expression refinement, and is defined below.

Algorithm 6 The $split(F, \omega, \Omega, \bar{\lambda})$ computes a set of downward refinements for the numerical facet restriction F targeting literals with label ω by exclusion of runs of literals with other labels from Ω relative to a set of most applicable contexts $\bar{\lambda}$.

```
1:  function split($F, \omega, \bar{\lambda}$)
2:      $F^{\mathcal{I}_{\bar{\lambda}}} = \{v_1, \ldots, v_n\}$          ▷ Ordered set of literals covered by $F$ in $\Delta^{\mathcal{D}}_{\bar{\lambda}}$
3:      $S := \emptyset$
4:      for all $\omega' \in \Omega$ where $\omega' \neq \omega$ do
5:          for all $\{v_i, \ldots, v_j\} \in contig_{max}(\omega, F^{\mathcal{I}_{\bar{\lambda}}})$ do
6:              if $i = 1$ and $j = n$ then
7:                  $low := \emptyset, high := \emptyset$          ▷ Entire set removed
8:              else if $i = 1$ and $j < n$ then
9:                  $low := \emptyset, high := \{v_{j+1}, \ldots, v_n\}$          ▷ LHS removed
10:             else if $i > 1$ and $j = n$ then
11:                 $low := \{v_1, \ldots, v_{i-1}\}, high := \emptyset$          ▷ RHS removed
12:             else if $i > 1$ and $j < n$ then
13:                 $low := \{v_1, \ldots, v_{i-1}\}, high := \{v_{j+1}, \ldots, v_n\}$          ▷ Split in two
14:             end if
15:             $S := S \cup rnr(F_i, low, high)$
16:         end for
17:     end for
18:     return $S$
19: end function
```

The function $rnr(F, low, high)$ takes an existing facet restriction expression F where $F^{\mathcal{I}_{\bar{\lambda}}} = \{v_1, \ldots, v_n\}$ and a new lower and upper target literal set $low = \{v_1, \ldots, v_{i-1}\}$ and $high = \{v_{j+1}, \ldots, v_n\}$ as computed by $split(F, \omega, \Omega, \bar{\lambda})$ to produce a set with the new facet restriction F' where $(F')^{\mathcal{J}_{\bar{\lambda}}} = low \cup high$, $v_0 < v_1$ and $v_n < v_{n+1}$ as follows:

$$rnr = \begin{cases} \emptyset & \text{if } low = \emptyset \wedge high = \emptyset \\ \emptyset & \text{if } low \cup high = F_i^{\mathcal{J}_{\bar{\lambda}}} \\ \{> v_j\} & \text{if } F_i = (> v_0) \wedge low = \emptyset \\ \{> v_0 \wedge < v_i\} & \text{if } F_i = (> v_0) \wedge high = \emptyset \\ \{(> v_0 \wedge < v_i) \vee (> v_j)\} & \text{if } F_i = (> v_0) \\ \{> v_j\} & \text{if } F_i = (\geq v_1) \wedge low = \emptyset \\ \{\geq v_1 \wedge < v_i\} & \text{if } F_i = (\geq v_1) \wedge high = \emptyset \\ \{(\geq v_1 \wedge < v_i) \vee (> v_j)\} & \text{if } F_i = (\geq v_1) \\ \{> v_j \wedge < v_{n+1}\} & \text{if } F_i = (< v_{n+1}) \wedge low = \emptyset \\ \{< v_i\} & \text{if } F_i = (< v_{n+1}) \wedge high = \emptyset \\ \{(< v_i) \vee (> v_j \wedge < v_{n+1})\} & \text{if } F_i = (< v_{n+1}) \\ \{> v_j \wedge \leq v_n\} & \text{if } F_i = (\leq v_n) \wedge low = \emptyset \\ \{< v_i\} & \text{if } F_i = (\leq v_n) \wedge high = \emptyset \\ \{(< v_i) \vee (> v_j \wedge \leq v_n)\} & \text{if } F_i = (\leq v_n) \\ \{> v_j \wedge < v_{n+1}\} & \text{if } F_i = (> v_0 \wedge < v_{n+1}) \wedge low = \emptyset \\ \{> v_0 \wedge < v_i\} & \text{if } F_i = (> v_0 \wedge < v_{n+1}) \wedge high = \emptyset \\ \{(> v_0 \wedge < v_i) \vee (> v_j \wedge < v_{n+1})\} & \text{if } F_i = (> v_0 \wedge < v_{n+1}) \\ \{> v_j \wedge \leq v_n\} & \text{if } F_i = (> v_0 \wedge \leq v_n) \wedge low = \emptyset \\ \{> v_0 \wedge < v_i\} & \text{if } F_i = (> v_0 \wedge \leq v_n) \wedge high = \emptyset \\ \{(> v_0 \wedge < v_i) \vee (> v_j \wedge \leq v_n)\} & \text{if } F_i = (> v_0 \wedge \leq v_n) \\ \{> v_j \wedge < v_{n+1}\} & \text{if } F_i = (\geq v_1 \wedge < v_{n+1}) \wedge low = \emptyset \\ \{\geq v_1 \wedge < v_i\} & \text{if } F_i = (\geq v_1 \wedge < v_{n+1}) \wedge high = \emptyset \\ \{(\geq v_1 \wedge < v_i) \vee (> v_j \wedge < v_{n+1})\} & \text{if } F_i = (\geq v_1 \wedge < v_{n+1}) \\ \{> v_j \wedge \leq v_n\} & \text{if } F_i = (\geq v_1 \wedge \leq v_n) \wedge low = \emptyset \\ \{\geq v_1 \wedge < v_i\} & \text{if } F_i = (\geq v_1 \wedge \leq v_n) \wedge high = \emptyset \\ \{(\geq v_1 \wedge < v_i) \vee (> v_j \wedge \leq v_n)\} & \text{if } F_i = (\geq v_1 \wedge \leq v_n) \end{cases}$$

We can now define how numerical facet restrictions N over the integers $\mathbb{Z}$ and doubles $\mathbb{R}$ are refined with the downward refinement operator $\rho_{\bar{\lambda}}$, as follows:

$$\rho_{\bar{\lambda}}(N) = \begin{cases} \{N'_\omega \mid \forall \omega \in \Omega, N'_\omega \in \mathit{mgnfr}(\omega, \bar{\lambda})\} & \text{if } N = \top^{\mathbb{Z}} \text{ or } N = \top^{\mathbb{R}} \\ \{t[F_1 \vee \ldots \vee F'_i \vee \ldots \vee F_n] \mid \forall i \in \{1, \ldots, n\}, & \text{if } N_\omega = t[F_1 \vee \ldots \vee F_n] \\ \quad \forall F'_i \in \mathit{split}(F_i, \omega, \Omega, \bar{\lambda})\} & \quad \text{where } n \geq 1 \end{cases}$$

Note that the definition of $\rho_{\bar{\lambda}}(N)$ for numerical facet restriction expressions N relies on functions *mgnfr* and *split*. These two functions require the computation of the approximate local domain of ordered numerical literals $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ every time they are invoked. For knowledge bases containing a large amount of numerical literal data, this may be expensive. However, the knowledge bases we consider will often contain fewer data literals than abstract individuals, as we will show in Chapter 6. In these cases, this method performs efficiently given that sets $\Delta^{\mathcal{D}}_{\bar{\lambda}}$ are restricted in size for any set of contexts $\bar{\lambda}$, and that the functions *mgnfr* and *split* are both efficient in that they have computational complexities which are at most PTIME.

Finally, we include the following rules for downward refining boolean literals $\mathbb{B}$ and string literals $\mathbb{S}$ which begin refinement from the top datatype in either domain:

$$\rho_{\bar{\lambda}}(N) = \begin{cases} \{\mathit{boolean}[= v] \mid \forall v \in \Delta^{\mathbb{B}}_{\bar{\lambda}}\} & \text{if } N = \top^{\mathbb{B}} \\ \{\mathit{string}[= v] \mid \forall v \in \Delta^{\mathbb{S}}_{\bar{\lambda}}\} & \text{if } N = \top^{\mathbb{S}} \end{cases}$$

These downward refinement rules simply generate $t[= v]$ facet restrictions to single out individual values in each domain. While this suffices for boolean values as there are only two, namely $\mathit{boolean}[= \mathit{true}]$ and $\mathit{boolean}[= \mathit{false}]$, clearly more complex rules can be defined over the set of strings, however we do not consider these.

4.5 An Improved Downward Refinement Operator

Throughout the previous section, we constructed the downward refinement operator $\rho_{\bar{\lambda}}$ in a piecewise way to address each of the cases for refinement, namely:

- Atomic and negated atomic concept conjunctions (§4.3.1);
- Role expressions (§4.3.2);
- Disjunctions (§4.3.3);
- Numerical, boolean and string-based concrete domains (§4.4).

The full definition for the operator $\rho_{\bar{\lambda}}$ is given in Definition 4.5.1.

Definition 4.5.1. *Given a concept expression C or datatype facet restriction N and a set of most applicable contexts $\bar{\lambda}$, the downward refinement operator $\rho_{\bar{\lambda}}(C)$ is defined as follows:*

$$\rho_{\bar{\lambda}}(C) = \begin{cases}
\{C_1 \sqcup \ldots \sqcup C_n \mid C_{1 \leq i \leq n} \in mga(\bar{\lambda}) & \text{if } C = \top \\
\quad \forall C_{1 \leq i < j \leq n} : C_i \preceq C_j\} & \\
\{A \mid A \in arf(\bar{\lambda}), A \sqsubset_{\mathcal{J}_{\bar{\lambda}}} C, \neg\exists A' \in arf(\bar{\lambda}) \text{ s.t.} & \text{if } C \in arf(\bar{\lambda}) \\
\quad A \sqsubset_{\mathcal{J}_{\bar{\lambda}}} A' \sqsubset_{\mathcal{J}_{\bar{\lambda}}} C\} \cup \{C \sqcap A \mid C \sqcap A \in arf(\bar{\lambda}), & \\
\qquad C \preceq A, C \text{ overlaps } A\} \cup & \\
\{C \sqcap \Diamond r.D \mid \forall(\Diamond r.D) \in ir(\bar{\lambda}) \text{ s.t. } C \preceq (\Diamond r.D)\} & \\
\{C_1 \sqcap \ldots \sqcap C_n \sqcap A \mid A \in arf(\bar{\lambda}), & \text{if } C = C_1 \sqcap \ldots \sqcap C_n \\
\quad C_n \preceq A, \forall C_{1 \leq i \leq n} \forall \lambda \in \bar{\lambda} : C_i \in arf(\bar{\lambda}) \rightarrow & (n \geq 2) \\
\qquad C_i \text{ overlaps } A\} \cup & \\
\{C_1 \sqcap \ldots \sqcap C_n \sqcap \Diamond r.D \mid \forall(\Diamond r.D) \in ir(\bar{\lambda}) \text{ s.t.} & \\
\quad C_n \preceq (\Diamond r.D)\} & \\
\{C_1 \sqcap \ldots \sqcap \Diamond s.E \sqcap \ldots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \ldots \sqcap \\
\quad \forall(\Diamond s.E) \in \rho_{\bar{\lambda}}(\Box r.D) \text{ s.t.} & \Box r.D \sqcap C_i \sqcap \\
\qquad \forall C_{i \leq j \leq n} \Diamond s.E \preceq C_j \wedge & \ldots \sqcap C_n \ (i \leq n) \\
\qquad\quad ncu(C_1 \sqcap \ldots \sqcap \Diamond s.E \sqcap \ldots \sqcap C_n)\} & \\
\{\exists r.E \mid E \in \rho_{\bar{\lambda}'}(D)\} \cup \{\exists s.(D) \mid s \in sh_{\downarrow}(r)\} \cup & \text{if } C = \exists r.D \\
\quad \{\geqslant^n r.(D) \mid \geqslant^n r.(D) \in re(\bar{\lambda}) \wedge n \geq 2 \wedge & \\
\quad \neg\exists(\geqslant^m r.(D)) \in re(\bar{\lambda}) \text{ s.t. } m < n \wedge m \geq 2\} & \\
\{\forall r.E \mid E \in \rho_{\bar{\lambda}'}(D)\} \cup \{\forall s.(D) \mid s \in sh_{\downarrow}(r)\} & \text{if } C = \forall r.D \\
\{\geqslant^n r.D \mid \geqslant^n r.D \in re(\bar{\lambda}) \wedge n > m \wedge & \text{if } C = \geqslant^m r.D \\
\quad \neg\exists(\geqslant^o r.(D)) \in re(\bar{\lambda}) \text{ s.t. } m < o < n\} \cup & \\
\{\geqslant^m s.D | s \in sh_{\downarrow}(r)\} \cup \{\geqslant^m r.E \mid E \in \rho_{\bar{\lambda}'}(D)\} & \\
\{\leqslant^n r.D \mid \leqslant^n r.D \in re(\bar{\lambda}) \wedge n < m \wedge & \text{if } C = \leqslant^m r.D \\
\quad \neg\exists(\leqslant^o r.(D)) \in re(\bar{\lambda}) \text{ s.t. } n < o < m\} \cup & \\
\{\leqslant^m s.D | s \in sh_{\downarrow}(r)\} \cup \{\leqslant^m r.E \mid E \in v_{\bar{\lambda}'}(D)\} & \\
\{C_1 \sqcup \ldots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \ldots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \ldots \sqcup C_n \\
\quad D \in \rho_{\bar{\lambda}}(C_i) \wedge C_{i-1} \preceq D \preceq C_{i+1}, 1 \leq i \leq n\} & (n \geq 2)
\end{cases}$$

$$\rho_{\bar{\lambda}}(N) = \begin{cases} \{N'_\omega \mid \forall\omega \in \Omega, N'_\omega \in mgnfr(\omega, \bar{\lambda})\} & \textit{if } N = \top^{\mathbb{Z}} \textit{ or } N = \top^{\mathbb{R}} \\ \{t[F_1 \vee \ldots \vee F'_i \vee \ldots \vee F_n] \mid \forall i \in \{1, \ldots, n\}, & \textit{if } N_\omega = t[F_1 \vee \ldots \vee F_n] \\ \quad \forall F'_i \in split(F_i, \omega, \Omega, \bar{\lambda})\} & \quad \textit{where } n \geq 1 \\ \{boolean[= v] \mid \forall v \in \Delta^{\mathbb{B}}_{\bar{\lambda}}\} & \textit{if } N = \top^{\mathbb{B}} \\ \{string[= v] \mid \forall v \in \Delta^{\mathbb{S}}_{\bar{\lambda}}\} & \textit{if } N = \top^{\mathbb{S}} \end{cases}$$

We now make some remarks about the theoretical properties of this refinement operator in terms of redundancy, completeness, and properness.

4.5.1 Properties of $\rho_{\bar{\lambda}}$

Proposition 4.5.2. *The downward refinement operator $\rho_{\bar{\lambda}}$ is not complete.*

Proof. It suffices to show that $\rho_{\bar{\lambda}}$ is not complete because refinements of maximum qualified cardinality restrictions such as ${}^{\leqslant n}r.D \rightsquigarrow_{\rho_{\bar{\lambda}}} {}^{\leqslant n}r.E$ where $D \sqsubseteq E$ are performed via the incomplete upward refinement operator $\upsilon_{\bar{\lambda}}$ (Definition 4.3.6) where $E \in \upsilon_{\bar{\lambda}}(D)$. □

Proposition 4.5.3. *The downward refinement operator $\rho_{\bar{\lambda}}$ is redundant.*

Proof. It suffices to show that $\rho_{\bar{\lambda}}$ will generate redundant refinement chains as shown in Example 4.3.3. □

Proposition 4.5.4. *The downward refinement operator $\rho_{\bar{\lambda}}$ is not proper.*

Proof. It suffices to show that $\rho_{\bar{\lambda}}$ may introduce disjunctions of the form $A \sqcup A$ for some concept A, as $A \sqcup A \equiv A$. □

Despite the operator $\rho_{\bar{\lambda}}$ being not complete, redundant and not proper, each of these characteristics has been carefully considered in the design of $\rho_{\bar{\lambda}}$ (§4.2.4) to ensure that it only avoids the construction of clearly unsatisfiable concept expressions, or those with an empty closed-world interpretation, for the purposes of efficiency.

As we will present in Chapter 6, we are able to integrate $\rho_{\bar{\lambda}}$ into learning algorithms which prove to be quite efficient and achieve strong results which outperform the state-of-the-art with respect to systems like DL-Learner which employ the relatively simple yet complete refinement operator ρ_B.

As is the case with the DL-Learner system, the inefficiency which results from the redundancy and improperness of the refinement operators ρ_B and $\rho_{\bar{\lambda}}$ can be handled not by modifying the refinement operator but by detecting redundancy post-refinement for any concept, such as by checking for duplicate concepts in a search beam, or a global set of concepts which a search has considered before, at the cost of extra memory usage at runtime, which we consider in subsequent chapters.

4.6 Summary

In this chapter, we have presented an analysis of a state-of-the-art refinement operator for the highly expressive DL known as $\mathcal{SROIQ}$ and identified several inefficiencies which we addressed with the definition of a new operator (§4.1). We then presented novel work around a method of defining the closed-world interpretation of subexpressions of any concept expression known as a context-specific interpretation (§4.2). We then showed how an approximation of a context-specific interpretation can be computed and then usefully applied to the definition of a new refinement operator (§4.3) which mitigates the issues identified with the state-of-the-art operator in Section 4.1, and introduces novel method for extending the downward refinement operator for learning over concrete domains with a view to supporting classification and subgroup discovery (§4.4). In Chapter 5, we will describe how we integrate the refinement operator known as $\rho_{\bar{\lambda}}$ into learning algorithms for classification and subgroup discovery, before analysing their performance in Chapter 6.

Chapter 5

Supervised Learning in DL Knowledge Bases

Supervised learning over DL knowledge bases involves searching for concepts which cover sets of labelled *example* individuals which meet some quality criteria. As we saw in Section 3.3.1 of Chapter 3, two examples of supervised learning are *classification* and *subgroup discovery* which will be the main focus of this chapter. Although the supervised learning methods presented in this chapter are applicable to learning problems involving two or more example labels, we will often restrict our attention to *binary* learning problems where any example e from the set of all examples $\mathcal{E}$ is labelled with $\omega \in \{+,-\}$, either *positive* (+) or *negative* (-) for ease of exposition without loss of generality.

Throughout this chapter, we address the problems of classification and subgroup discovery with application of the refinement operator $\rho_{\bar{\lambda}}$ as developed in Section 4.1 of Chapter 4. We will show how this operator can be incorporated into generate-and-test learning algorithms (§3.4.1) and how the auxiliary data structures which $\rho_{\bar{\lambda}}$ depends on, such as the context graph (§4.2.2) and local domains (§4.2), support the efficient evaluation of concepts with these algorithms. In applying $\rho_{\bar{\lambda}}$ to search concepts, we will also address how the properties of *redundancy* and *improperness* may adversely affect the performance of the learning algorithms we present (§5.3), and describe methods to mitigate these problems.

5.1 Supervised Learning

Given a set of example data $\mathcal{E}$ where each example $e \in \mathcal{E}$ is labelled with one of $\omega \in \{+,-\}$, *supervised learning* generally seeks to find concepts which cover a set of examples to represent a certain distribution of labels. In the problem of *subgroup dis-*

covery, concepts are sought, also known as *candidates* or *hypotheses* h, which describe a set of examples with an *unusual* distribution relative to the set of all examples. Consider the case where we have 100 examples labelled $+$ and 100 labelled $-$, giving us a 50/50 split of examples labelled $+$ to $-$. If a hypothesis concept h covers 90 examples labelled $+$ and 10 examples labelled $-$, the split becomes 90/10 and may be considered sufficiently unusual compared to 50/50 such that h is deemed to be *interesting*. Interestingness is usually defined in terms of a threshold on a *correlation measure* function which assesses the proportions of labelled examples which are deemed significant enough for some hypothesis h to be considered a *solution* to a learning problem. For example, in subgroup discovery, weighted relative accuracy (Definition 3.3.4) is one such function which can be used in this way. Generally, hypotheses h found to be interesting are *descriptive* in that the structure of h reveals what it is about the examples it covers that results in an unusual distribution in the cover. Subgroup discovery therefore generalises many *data mining* tasks which seek hypotheses which describe interesting clusters or groups of labelled examples. The very structure of a hypothesis h directly reveals *why* examples are covered, which ought to provide insight into the problem being solved. Hypotheses posed as DL concepts are particularly suited for this task, as they can be composed from domain-specific human comprehensible terms such as class and property names from an OWL ontology which describes, for example, terms from medicine, genomics, or manufacturing.

The problem of *classification* can be seen as a special case of subgroup discovery where hypothesis concepts h are sought which cover *only* those examples labelled with certain labels to the exclusion of all others, such as when h covers all examples labelled $+$ and no examples labelled $-$. In this way, hypotheses sought for solving classification problems are often intended for use in *prediction*, as given a previously unseen example e, we may provide a label for e by checking if it may be an instance of h in which case it is labelled $+$, otherwise $-$. Therefore, in classification, we seek concepts h which *generalise* well to unseen examples such that it performs well in prediction tasks.

We will now describe a generalised search strategy which employs the use of the refinement operator $\rho_{\bar{\lambda}}$ for searching a space of concepts which is appropriate for both subgroup discovery and the more specialised problem of classification (§5.1.1). In the definition of our search algorithm, we will analyse various measure functions used for assessing the performance of concepts and show how they may indicate when parts of the search space may be pruned to improve performance (§5.1.1.2).

5.1.1 Learning as Search

In Chapter 4, we developed the downward refinement operator $\rho_{\bar{\lambda}}$ to control the set of concepts refined from any concept expression C such that it avoids the production of many concepts which are clearly unsatisfiable or redundant, with the aim of reducing the search space of concepts. While $\rho_{\bar{\lambda}}$ may exclude concepts which are of little value when used to solve learning problems by generate-and-test methods, the resulting space of concepts which $\rho_{\bar{\lambda}}$ structures and operates over may still be impractically large for knowledge bases containing a large number of concepts and roles. Therefore, simple methods which enumerate the entire space of concepts such as Algorithm 2 cannot be used in practice, and we are instead required to bound the total number of hypotheses under consideration at any stage of the algorithm to limit memory usage.

Algorithm 3 of Chapter 3 presented what is known as a basic *beam search* strategy appropriate for searching through a large space of candidate concept expressions. In this algorithm, the set of concepts under consideration at any one time is bounded with a maximum cardinality, and only the candidates deemed to be the best are maintained at any point in the algorithm. While this approach effectively addresses the problem of managing a large search space, it results in the learning algorithm being *approximate* as solutions which are refinements of candidates which were excluded from the beam set at any point will not be explored. Beam search methods rely on the use of so-called *heuristic* functions to determine which candidates ought to be maintained in the beam with the expectation that solutions are to be found amongst their refinements.

Heuristic functions are often defined in terms of the current performance of a candidate concept h relative to the learning problem being solved, along with other measures such as the structural complexity of h. For example, in a classification problem, the *accuracy* function may be used to assess current performance, while structural complexity can be measured simply by the number of terms in the expression as $length(h)$. These functions can be combined into one heuristic *utility* function $\mathbf{u}$ as per Definition 3.4.12 as follows:

$$\mathbf{u}(h, \mathcal{E}) = acc(h, \mathcal{E}) - \beta \cdot length(h)$$

where *length* : $\mathcal{L} \mapsto \mathbb{N}$ maps concept expressions to their length as the number of terms in the expression $h \in \mathcal{L}$, and where $\beta \in [0,1]$ is a fixed parameter which captures the user-defined importance of concept length relative to accuracy, and is chosen experimentally. By penalising the utility of a concept based on its length, we are preferring shorter concepts with high accuracy over longer ones, which embodies the well-known *Minimum Description Length (MDL) principle* [84], whereby it is anticipated that simpler, shorter hypotheses ought to generalise better than longer ones over unseen data for the purposes of prediction.

Accuracy gain is another technique which may be used in defining a heuristic function for use in concept learning which is to incorporate the degree to which accuracy increased or decreased from some concept h_0 to one of its refinements h_1, reflecting the intuition that relatively large stepwise gains or losses in accuracy are indicative of heading closer to or further away from solutions from $h_0 \rightsquigarrow h_1$. This can be incorporated into a utility function as follows:

$$\mathbf{u}(h_1, \mathcal{E}) = acc(h_1, \mathcal{E}) + \alpha \cdot (acc(h_1, \mathcal{E}) - acc(h_0, \mathcal{E})) - \beta \cdot length(h_1)$$

The fixed parameter $\alpha \in [0,1]$ is similar to β in that it determines the user-defined importance of accuracy gain (or loss) relative to the accuracy of h. This utility function is used by the DL-Learner system for classification in the CELOE and OCEL search strategies, except that instead of $length(h)$, the latter employs the length of the refinement chain $\top \rightsquigarrow \ldots \rightsquigarrow h$ instead [58].

Recall that a utility function $\mathbf{u}$ induces a *ranking* over concepts where, for two concepts C, D and a set of examples $\mathcal{E}$, that $\mathbf{u}(C, \mathcal{E}) < \mathbf{u}(D, \mathcal{E})$ implies that D is preferred over, or is *stronger* than, concept C. In a search algorithm which selects candidate expressions such as C or D to refine towards concepts which are solutions, the utility function can be used to describe which concepts are the best to refine first over any others. For example, in a beam search with a beam set B of cardinality n for which all members have been refined into a set $E = \{h \mid \forall b \in B : h \in \rho(b)\}$, repopulation of the beam B may be performed by selecting the n-best candidates of E according to their utility $\mathbf{u}$.

In addition to a utility function, a search algorithm will impose a minimum threshold τ on a measure function to define a boolean *quality* function $\mathcal{Q}$ which is used to indicate when a hypothesis h may be considered a solution to a learning problem. For example, hypotheses h which can be considered solutions where accuracy is used as the measure function may threshold accuracy where $\tau = 0.95$ as follows:

$$\mathcal{Q}(h, \mathcal{E}) = acc(h, \mathcal{E}) > 0.95$$

A search algorithm which assesses the performance of hypotheses based on a measure function f may also be able to leverage certain properties of f to determine if refinements of any particular candidate hypothesis h could *ever* satisfy the quality function. Such properties of the measure functions are those of *anti-monotonicity* or *convexity*, as we will describe in the next two sections. These properties permit us to define lower bounds on the coverage of certain sets of labelled examples for any hypothesis h which indicate if any of the refinements $\rho^*(h)$ could ever be considered a solution. If they cannot, the space of concepts defined by the set $\rho^*(h)$ may be effectively pruned from the search space, thus improving efficiency. In the next two sections, we will analyse these properties against several common measure functions.

5.1.1.1 Anti-Monotonic Quality Criteria

Recall that a boolean quality function $\mathcal{Q}$ is *anti-monotonic* by Definition 3.4.9 if, for any two concepts C, D where $C \sqsubseteq D$, that if $\mathcal{Q}(C)$ succeeds, $\mathcal{Q}(D)$ necessarily succeeds. If the quality function $\mathcal{Q}$ is defined in terms of a threshold τ on a measure function $f(C, \mathcal{E}) \geq \tau$ which fails for any downward refinement $E \in \rho^*(C)$, we conclude that the condition $f(h, \mathcal{E}) \geq \tau$ is anti-monotonic for any concept h.

The *relFreq* function which computes *relative frequency* of Definition 3.4.10 is an example of an anti-monotonic measure function which we can use to apply to a search. Basically, *relFreq* can be used to ensure that any candidate concept C must strictly cover a minimum proportion of examples from all examples $\mathcal{E}$ as:

$$relFreq(C, \mathcal{E}) \geq \tau_{min}$$

where $\tau_{min} \in [0, 1]$ and represents a minimum threshold on relative frequency, for example, $\tau_{min} = 0.1$ requires that any candidate C must cover at least 10% of all examples. Clearly, this condition is anti-monotonic as any generalisation D where $C \sqsubseteq D$ will also have $relFreq(D, \mathcal{E}) \geq \tau_{min}$ as D cannot cover any fewer examples than C. This condition can be used for pruning the search space, as if we find that $relFreq(C, \mathcal{E}) < \tau_{min}$, then for all concepts $E \in\in \rho^*(C)$ we will have $relFreq(E, \mathcal{E}) < \tau_{min}$.

Other measures of quality, such as the accuracy function, are *not* anti-monotonic, and therefore, cannot be used to prune away parts of the search space based on simply thresholding on their values. Fortunately, we may still define similar conditions under which refinements of any concept may never be considered of sufficient quality if the measure function has the property of *convexity*, in which case we may impose certain minimum bounds on the cover of hypotheses such that we ensure that their refinements may still contain a candidate of sufficient quality.

5.1.1.2 Convex Quality Criteria

Generally, if a quality function $\mathcal{Q}$ is defined over a measure function f which can be shown to be *convex*, we may conclude that f is anti-monotonic which will assist us in understanding when to prune parts of a search space away to improve efficiency. To define the convexity of a function such as f, we first define the notion of a *convex set*, as follows.

Definition 5.1.1. ***(Convex Set)*** *A set S in some vector space such as $\mathbb{R}^d$ is* ***convex*** *if, for any two vectors $s_1, s_2 \in S$ and any $t \in [0,1]$, then the vectors described by $(1-t)s_1 + ts_2$ must also belong to S. Intuitively, this means that all vectors which lie on the straight line between any two vectors s_1 and s_2 appear in S.*

Definition 5.1.2. ***(Convex, Concave Function)*** *A function $f : S \mapsto \mathbb{R}$ is* ***convex*** *iff $S \subseteq \mathbb{R}^d$ is a convex set and $\forall s_1, s_2 \in S, t \in [0,1] : f(ts_1 + (1-t)s_2) \leq tf(s_1) + (1-t)f(s_2)$. If f is convex, then we say that $-f$ is* ***concave****, and vice-versa.* *[85]*

In order to analyse such functions f in terms of the covers of the concepts, we consider the *stamp point* function $sp : \mathcal{L} \times S \mapsto \mathbb{Z}^d$ which maps concepts $C \in \mathcal{L}$ and the *population set* of examples $\mathcal{E} \in S$ where each example $e \in \mathcal{E}$ is labelled with one of d distinct labels as e_{ω_i} where $\omega_i \in \Omega$ and $d = |\Omega|$ and $1 \leq i \leq d$ as:

$$sp(C, \mathcal{E}) = \langle x_{\omega_1}, \ldots, x_{\omega_d} \rangle$$

where $x_{\omega_i} = |C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^{\omega_i}|$. The function *sp* maps the *example cover* of some concept expression C to a so-called *stamp point* $\langle x_{\omega_1}, \ldots, x_{\omega_d} \rangle$ where each component x_{ω_i} represents the number of examples labelled ω_i for each label $1 \leq i \leq d$ in d-dimensional *coverage space* [65, 33]. Measure functions f can then be redefined in terms of functions in coverage space with σ_f where:

$$\sigma_f(sp(C, \mathcal{E})) = f(C, \mathcal{E})$$

For example, consider the accuracy function as per Definition 3.3.3 where $\Omega = \{+,-\}$ which is mapped into coverage space as follows:

$$sp(C,\mathcal{E}) = \langle x,y\rangle \text{ where } x = TP = |C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^{+}|, \text{ and } y = FP = |C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^{-}|$$

where x describes the number of examples labelled $+$ and y describes the number of examples labelled $-$ in the cover of C. In terms of two-dimensional coverage space, the accuracy function $acc(C,\mathcal{E})$ where $P = TP + FN = |\mathcal{E}^{+}|$ and $N = FP + TN = |\mathcal{E}^{-}|$ becomes:

$$\sigma_{acc}(\langle x,y\rangle) = \frac{x + (N-y)}{x + y + (P-x) + (N-y)} = \frac{x+N-y}{P+N}$$

Consider the case where any concept C is considered a solution to a learning problem by imposing a minimum threshold τ_{min} over accuracy as $\mathcal{Q}(C,\mathcal{E}) = acc(C,\mathcal{E}) \geq \tau_{min}$. By rearranging for y, the definition of accuracy in coverage space becomes:

$$y \leq x - \tau_{min}(N+P) + N$$

By this equation, concept C meets the criteria to be considered a solution given the numbers x, y of covered examples labelled $+,-$ when this equation is satisfied. In Figure 5.1, we plot this as a function in coverage space for various values of minimum accuracy τ_{min}, which each represent *isometric lines* passing through all points $\langle x,y\rangle$ for a fixed value of τ_{min}. Here, we see that the isometric lines of the accuracy function are linear, which is the precise condition under which a function may be both convex and concave.

From Figure 5.1 we see the isometric lines for the accuracy function plotted for both $\tau_{min} = 0.75$ and $\tau_{min} = 0.95$, where those candidate concepts with covers $\langle x,y\rangle$ which lie in the area between each respective isometric line and the x-axis representing solutions which meet or exceed the minimum accuracy τ_{min}.

The point at which isometric lines in coverage space cross an axis such as where $y = 0$ provides us with *lower bounds* on the other variables, such as x. In this example, the isometric line for $\tau_{min} = 0.75$ crosses the x-axis at $x = 50$ when $y = 0$, therefore no candidate with a stamp point $\langle x,y\rangle$ can ever be a solution where $\sigma_{acc}(\langle x,y\rangle) \geq 0.75$ when $x < 50$. This analysis provides us with conditions with which we may use to prune candidates and all of their specialisations from a downward-refinement search, as concept covers $\langle x,y\rangle$ may only ever be *reduced*.

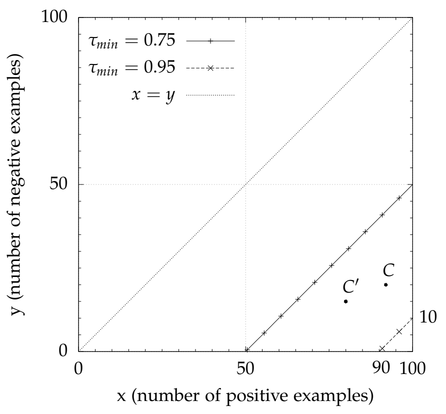


Figure 5.1: A *coverage space* plot representing the number of positive examples labelled $+$ on the x-axis and negative examples labelled $-$ on the y-axis. Two *isometric lines* for accuracy threshold values $\tau_{min} \in \{0.75, 0.95\}$ for a problem with 100 positive and 100 negative examples are shown, along with the diagonal line at $x = y$. Also plotted are the *stamp points* of two hypotheses, C at $\langle 92, 20\rangle$ and C' at $\langle 80, 15\rangle$.

Given any candidate h with stamp point $\langle x, y\rangle$ in coverage space, we can also define *upper bounds* on the potential future value of any downward refinement of h by inspecting the value of the measure function σ_f by setting one of the variables of the stamp point to 0. For example, consider the two candidates C at stamp point $\langle 92, 20\rangle$ and C' at point $\langle 80, 15\rangle$ from Figure 5.1. The upper bounds on the potential future value of accuracy for any number of refinements of C or C' is found by assuming the refinements have covers which contain no negative examples, as follows:

- $C : \sigma_{acc}(\langle 92, 0\rangle) = 0.96$
- $C' : \sigma_{acc}(\langle 80, 0\rangle) = 0.9$

With respect to the minimum threshold $\tau_{min} = 0.95$, we find downward refinements of C can *potentially* reach a refinement with accuracy 0.96, therefore should be considered for future refinement. However, as C' has an upper bound of 0.9, we may safely prune it from the search as C' and all of its downward refinements $\rho^*(C')$ can never satisfy $\sigma_{acc} \geq 0.95$. The notion of computing an upper bound on the future potential value of convex measure functions σ_f was formalised by Zimmerman and De Raedt [111] as follows:

Definition 5.1.3. *(**Upper Bounds on Convex Measure Functions**) The upper bound on values of a convex measure function σ_f with respect to a candidate hypothesis h with stamp point $\langle x, y\rangle$ is given by the function ub_{σ_f} as:*

$$ub_{\sigma_f}(\langle x, y\rangle) = max\ \{\sigma_f(\langle x, 0\rangle), \sigma_f(\langle 0, y\rangle)\}$$

Upper bounds on the potential future value of any candidate computed with ub_{σ_f} permit us to prune candidates from a search based directly on a minimum threshold τ_{min} on σ_f.

While accuracy is useful for binary classification problems, it is not useful for subgroup discovery in general which aims to locate hypotheses which cover an unusual distribution of examples relative to a population, such as hypotheses which cover more examples labelled $-$ than labelled $+$. In these problems, so-called *correlation measures* are often used to assess the interestingness of a candidate to define which hypotheses are solutions. Two common correlation measures for assessing the interestingness of subgroups are the χ^2 statistic, and *weighted relative accuracy* (WRA), which have both been shown to be convex functions [65, 111].

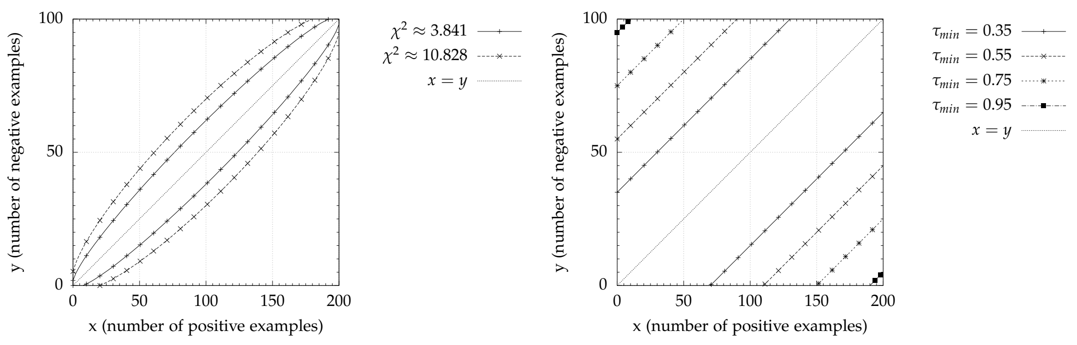


Figure 5.2: Plots of various isometric lines in coverage space for χ^2 on the left, and weighted relative accuracy (WRA) on the right for a population of 200 examples labelled $+$ on the x-axis, and 100 examples labelled $-$ on the y-axis.

Figure 5.2 shows various isometric lines for χ^2 and WRA, which lie either side of the diagonal $x = y$ from $(0, 0)$ to (P, N). Note that the isometric lines for various threshold values are reflected on either side of the diagonal to capture interestingness irrespective of the proportion of examples labelled either with $+$ or $-$ relative to the population, which here is shown where $P = 200$ and $N = 100$. Candidates with coverages $\langle x, y\rangle$ may be considered solutions when their point in coverage space does lies outside the space between the two reflected isometric curves [32].

The χ^2 function is often used to evaluate the statistical significance of the coverage of candidates relative to the example population in rule learning systems. In Figure 5.2, χ^2 is shown to be thresholded at $\chi^2 \geq 3.841$ beyond which a candidate cover represents a statistically significant distribution of examples relative to the population with a probability of more than 95%, and where $\chi^2 \geq 10.828$, with a probability of more than 99.9%.

WRA is a frequently used correlation measure which has been shown to perform well in rule induction systems such as CN2 [54]. In Figure 5.2, various isometric lines are shown for various levels of significance of 35% to 95% relative accuracy. This function differs from the accuracy function in that it accounts for skewed proportions of examples with different labels, whereas accuracy relies of the distribution of examples between two labels $+$ and $-$ to be relatively even. Secondly, WRA may be used to determine the significance of a candidate's cover relative to a population of examples whether the cover is composed of examples with labels skewed towards one of $+$ versus $-$, whereas accuracy is intended to model the performance of covers of examples relative to one label only, $+$.

Another commonly used convex correlation measure used in binary classification is the Matthews correlation coefficient (MCC) [76] function, and is often used as an alternative to accuracy when the distribution of example labels is uneven. MCC is related to χ^2 where $|MCC| = \sqrt{\frac{\chi^2}{x+y}}$ which, when expressing in terms of x, y in two-dimensional coverage space is:

$$\sigma_{mcc}(\langle x,y\rangle) = \frac{x(N-y)-y(P-x)}{\sqrt{PN(x+y)(P+N-x-y)}}$$

By rearranging for y and thresholding on τ, we obtain the function for isometric lines of MCC in coverage space:

$$y = \frac{N^2P\tau^2 + NP^2\tau^2 \pm \sqrt{N}\sqrt{P}\tau(N+P)\sqrt{NP\tau^2+4Px-4x^2} - 2NP\tau^2x + 2NPx}{2(NP\tau^2+P^2)}$$

MCC evaluates to 1 when the candidate cover $\langle x,y\rangle = (P,0)$, -1 when $\langle x,y\rangle = (0,N)$ and 0 when $\langle x,y\rangle$ lies on the diagonal between $(0,0)$ and (P,N) representing little to no difference from the distribution of the example population.

In order to assess the significance of any candidate h with stamp point $\langle x,y\rangle$ relative to σ_{mcc}, we may impose a threshold τ_{min} on the absolute value as:

$$|\sigma_{mcc}(\langle x,y\rangle)| \geq \tau_{min}$$

With this description, we may impose upper bounds on the MCC measure function for any candidate stamp point $\langle x,y\rangle$ as follows:

$$ub_{\sigma_{mcc}}(\langle x,y\rangle) = max\left\{\left|\frac{-yP}{\sqrt{yPN(P+N-y)}}\right|, \left|\frac{xN}{\sqrt{xPN(P+N-x)}}\right|\right\}$$

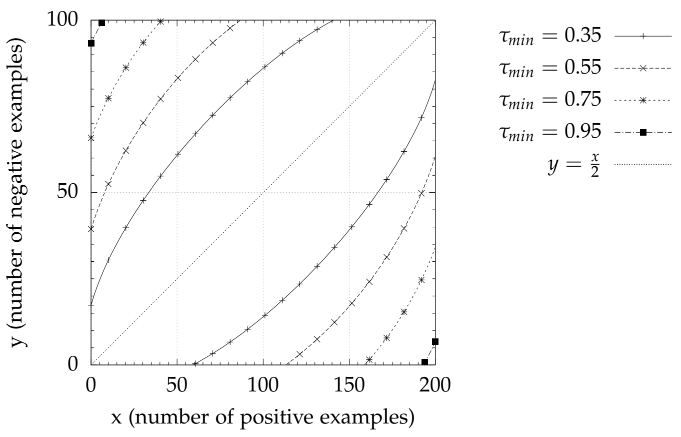


Figure 5.3: Various isometric lines for thresholded values of the *Matthews correlation coefficient (MCC)* in coverage space for a population of 200 examples labelled $+$ on the x-axis, and 100 examples labelled $-$ on the y-axis.

Example 5.1.4. *Consider a binary labelled set of examples* $\mathcal{E}$ *where* $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ *and where* $P = |\mathcal{E}^+| = 200$ *and* $N = |\mathcal{E}^-| = 100$*, with three candidate concepts C at stamp point* $\langle 160, 50 \rangle$*, D at* $\langle 150, 50 \rangle$ *and E at* $\langle 150, 70 \rangle$*. When the minimum MCC threshold* $\tau_{min} = 0.75$ *as shown in Figure 5.3 is used, the following upper bounds on* σ_{mcc} *apply:*

$$\begin{aligned} ub_{\sigma_{mcc}}(sp(C, \mathcal{E})) &= max\,\{0.632, 0.756\} &&= 0.756 \\ ub_{\sigma_{mcc}}(sp(D, \mathcal{E})) &= max\,\{0.632, 0.707\} &&= 0.707 \\ ub_{\sigma_{mcc}}(sp(E, \mathcal{E})) &= max\,\{0.780, 0.707\} &&= 0.780 \end{aligned}$$

Candidate D may be pruned as all of its downward refinements $D' \in \rho^*(D)$ *will never have a cover which permits* σ_{mcc} *to exceed the minimum threshold of* $\tau_{min} = 0.75$.

Note that the upper bound computed by ub_{σ_f} for some convex measure function f given a candidate h over examples $\mathcal{E}$ is *optimistic* in the sense that it is not guaranteed that some hypothesis h' will exist where $h' \in \rho^*(h)$ and $\sigma_f(sp(h', \mathcal{E})) = ub_{\sigma_f}(sp(h, \mathcal{E}))$. Instead, the upper bound only indicates the possibility of the existence of some h' which maximises σ_f to the upper bound value for the parent candidate h. Therefore, given a collection of candidates such as a beam set which is maintained in a beam search, we may order candidates based on their upper bounds in order to refine those which may lead to the strongest solutions. In this way, the upper bound of any candidate may be incorporated as a heuristic into a utility function $\mathbf{u}_{OM}$ to indicate the strength of potential solutions in the set of refinements of any candidate.

Definition 5.1.5. ***(Utility Function $\mathbf{u}_{\mathbf{OM}}$)*** *Given a concept d refined from c from the set of all concepts $\mathcal{L}$, a set of labelled examples $\mathcal{E}$, a convex measure function σ_f, the upper bound function ub_{σ_f}, the stamp point function sp, the length function to compute the number of symbols in any concept, and real-valued parameters $\alpha, \beta, \gamma \in [0,1]$, the utility function $\mathbf{u}_{OM}(d, \mathcal{E})$ is defined as:*

$$\begin{aligned}\mathbf{u}_{OM}(d,\mathcal{E}) = \;& |\sigma_f(sp(d,\mathcal{E}))| + \gamma \cdot ub_{\sigma_f}(sp(d,\mathcal{E})) + \\ & \alpha \cdot (|\sigma_f(sp(d,\mathcal{E}))| - |\sigma_f(sp(c,\mathcal{E}))|) - \beta \cdot length(d)\end{aligned}$$

The utility function $\mathbf{u}_{OM}$ of Definition 5.1.5 incorporates the following:

- The current performance of d according to the convex measure function σ_f in the range $[-1,1]$ where $|\sigma_f|$ reflect weak solutions at 0 and the strongest solutions at 1;
- The optimistic upper bound on σ_f for d;
- Any stepwise gain in the performance of d relative to its parent concept, c, according to σ_f;
- The complexity of the expression d as the number of symbols it contains.

Note that $\mathbf{u}_{OM}$ ranges over the set of reals, where larger values correspond to stronger candidates as per Definition 3.4.12. This utility function incorporates aspects of the OCEL and CELOE heuristics used in the DL-Learner system which capture gain in the measure function and penalise long concepts [58], but also incorporates the optimistic upper bound ub_{σ_f} on σ_f to capture the potential strength of candidates amongst the set of refinements. The user-defined parameter γ controls the importance of the optimistic upper bound of future potential of a candidate d. By incorporating this upper bound as a heuristic in the utility function, we boost the utility of currently low-performing candidates with high future potential, without lowering the utility of currently high-performing candidates also with high future potential. Experimentally, we have typically used the settings $\alpha = 0.5, \beta = 0.02, \gamma = 0.2$.

The use of the optimistic upper bound for candidate selection in a utility function for a search algorithm has been described before in the *cluster-grouping* (CG) algorithm [111]. This algorithm implements a best-first search which orders candidates for downward refinement exclusively on the value of their upper bound ub_{σ_f}. The CG algorithm then employs a pruning strategy which eliminates candidates and their refinements if it can be shown that their upper bound does not exceed a current minimum threshold τ which is initially defined by the user, but then is increased at runtime to match the weakest solution, if found. In a DL learning setting, concepts which are solutions may be very sparsely populated amongst the vast set of concepts in $\mathcal{L}$, which we estimate is larger than the hypothesis spaces over which the CG algorithm is designed to operate over. By ordering on upper bound values only, we risk exploring large parts of the search space based only on the potential future performance and may ignore currently high performing candidates which are close to solutions. Furthermore, we prefer small, simple concepts over longer ones which are semantically equivalent under a closed-world interpretation such as $(\mathcal{I}, \mathcal{U})$.

In a learning algorithm, once any solution C has been found which exceeds a minimum threshold τ_{min} for some measure function σ_f, we may opt to terminate the search, or continue to look for other concepts C' which exceed the performance of the last found solution C. So-called *anytime* algorithms work in this way, where the last best solution to the problem is maintained and is potentially improved given more computation time. If a solution is found with value $\sigma_f = \tau$ where $\tau \geq \tau_{min}$ and computation is allowed to continue to search for better solutions, the minimum bound τ_{min} may be reset to τ which has the effect of permitting the algorithm to prune more concepts from the search space. For example, consider a search algorithm which finds a solution with correlation measure value τ_{best}. Then, any subsequent candidate D under consideration which has an optimistic upper bound t where $t \leq \tau_{best}$ may be pruned, as none of the specialisations in $\rho^*(D)$ can have a correlation measure value which exceeds τ_{best}. Similarly, if a learning algorithm is designed to locate the top-k concepts for some fixed k, the threshold τ_{min} may simply be set to that of the weakest of the current set of j solutions where $1 \leq j \leq k$.

5.1.2 A Top-*k* Search Algorithm for Supervised Learning

In Section 3.4.3 of Chapter 3, we introduced the *beam search* Algorithm 3 for searching a space $\mathcal{L}$ of DL concepts by downward refinement. This algorithm traverses the space of concepts by maintaining a beam set B of maximum cardinality B_{max} to maintain the *search frontier*, namely, the set of all candidate concepts to consider for refinement at any one time. This algorithm proceeds by collecting refinements of the candidate concepts in B into a temporary *expansion set* E, before repopulating the beam set B with at most B_{max} of the best candidates from E, or by selecting candidates from E at random with a probability proportional to their performance relative to a utility function, a strategy known as *stochastic search*.

We aim to generalise this capability by permitting more control over the beam set B and expansion set E in a way which supports both memory-bounded *beam* and *best-first* search as follows. We introduce a maximum bound on the expansion set E as E_{max}, and maintain it by only permitting the best E_{max} candidates at any one time. When the algorithm has refined all candidates in the beam B populating the expansion set E, we permit control over how any *remaining* candidates in E are treated. In a beam search such as Algorithm 3, any remaining candidates in the expansion set E are discarded once the beam B is repopulated, as per line 4 of Algorithm 3. However, if we permit remaining candidates to reside in E, subsequent refinement of candidates of the beam B into E can be chosen on a best-first basis. This behaviour allows candidate concepts of varying lengths in the beam at any one time, and are selected based on the strength of their utility, which is a *best-first* search approach.

Furthermore, we will permit the search to maintain up to $k \geq 1$ of the best solutions found at any point, and limit the time the algorithm may spend searching for solutions with a maximum computation time t_{max} to support an *anytime* strategy. Our search method is shown as Algorithm 7, and is a combination of our modifications to the beam search of Algorithm 3 and the CG algorithm [111]. Our algorithm reflects the CG algorithm in that it dynamically increases the minimum bound τ on a measure function σ_f if any solutions are found such that candidates which have an upper bound $ub_{\sigma_f} < \tau$ may be pruned from the search space, as they can never be stronger than any of the current solutions. Additionally, any solution s which is subsumed by some pre-existing solution s' where $s \sqsubseteq s'$ which share the same stamp point $\langle x, y \rangle$ are excluded as they are unlikely to provide any additional information about the cover.

The main difference between Algorithm 7 and the CG algorithm is that the former permits bounds on the size of the set of candidates currently under consideration, as the set of candidates which have upper bounds on σ_f at any one time may be impractically large to store in memory. This is why the expansion set E is also limited with maximum size E_{max}, and when $|E| > E_{max}$, candidates with the worst upper bounds are pruned, as per line 24. Note that nodes with the weakest utility $\mathbf{u}_{OM}$ may also be used to prune candidates from E, especially if the utility function incorporates upper bounds on σ_f, as does $\mathbf{u}_{om}$.

Algorithm 7 is our general-purpose anytime search algorithm which is applicable for both supervised classification and subgroup discovery problems, and takes the following:

- $\mathcal{E}$: A set of binary labelled examples as $\mathcal{E} = \bigcup_{\omega \in \Omega} \mathcal{E}^{\omega}$ for $\Omega = \{+, -\}$;

- $\rho_{\bar{\lambda}}$: Our downward refinement operator defined against a concept language $\mathcal{L}$;
- k: The maximum number of concepts to find as solutions where $k \geq 1$;
- $\mathbf{u}_{OM}$: Our real-valued heuristic utility function;
- sp: The stamp point function mapping a concept C to a stamp point $\langle x, y\rangle$ in coverage space relative to $\mathcal{E}$ and Ω;
- σ_f: A convex measure function defined over coverage space;
- ub: The upper bound function defined over σ_f;
- τ_{min}: A minimum threshold on σ_f which signifies when candidates with stamp points $\langle x, y\rangle$ are solutions as $\sigma_f(\langle x, y\rangle) \geq \tau_{min}$;
- t_{max}: The maximum time for which the algorithm may execute;
- B_{max}: The maximum width of an *open/beam set* B where $|B| \leq B_{max}$ of candidates to refine, so as to bound memory consumption.
- E_{max}: The maximum width of an *expanded/successor set* E where $|E| \leq E_{max}$, also to bound memory consumption.
- *beam*: A boolean variable which, when $beam = true$, indicates that the search should use a *beam search* strategy similar to that of Algorithm 3 which rejects expanded candidates every time the beam is repopulated. Otherwise when $beam = false$, we revert to a *best-first search* strategy which constantly maintains at most E_{max} best candidates.

The function REPOPULATEBEAM as used on line 28 of Algorithm 7 regenerates the beam set B by selecting candidates from E. We describe two different implementations of this function, the first being Algorithm 8 which simply selects the best B_{max} candidates from E to repopulate B, and the second being Algorithm 9 which performs stochastic selection of the best B_{max} candidates from E with a probability proportional to candidate utility, and can be used to increase *diversity* in the search to avoid getting trapped in local maxima as illustrated in Figure 3.3.

Algorithm 9 computes the range of scores produced by $\mathbf{u}_{OM}$ for all candidates in the expansion set E to normalised all values into the range $[0, 1]$ where 0 represents the best utility and 1 represents the worst, as shown on line 18. With utilities normalised, the algorithm attributes each candidate with normalised utility n with a probability p proportional to $e^{-n/T}$, known as the Gibbs distribution, for some value of $T \in [0, 1]$. Candidates are then chosen at random according to this distribution until the beam set B is repopulated, or the set E is exhausted. The Gibbs distribution is commonly used in stochastic search methods as it permits control over the amount of diversity in the search with the *temperature* parameter T. For high values of T, the

Algorithm 7 Time- and Memory-Bounded Top-k Concept Search Algorithm

```
1: function TOP-k-SEARCH($\mathcal{E}, k, \sigma_f, \mathbf{u}_{OM}, sp, ub, \tau_{min}, t_{max}, B_{max}, E_{max}, beam$)
2:     $S := \emptyset$                                    ▷ Solution set
3:     $B := \{\top\}$                ▷ Open/beam set starting with the top concept $\top$
4:     $E := \emptyset$                                ▷ Expanded/successor set
5:     $\tau := \tau_{min}$              ▷ Initialise user-defined minimum quality on $\sigma_f$
6:     $t := t_0$                               ▷ Start computation timer at $t_0$
7:     while $(|B| > 0) \wedge (t < t_{max})$ do      ▷ Beam not empty and time not exceeded
8:         $h := h \in B$                              ▷ Arbitrary candidate hypothesis
9:         $B := B \setminus \{h\}$
10:        $C := \rho_{\bar{\lambda}}(h)$              ▷ Generate all single-step specialisations of $h$
11:        $E := E \cup \{c \in C \mid \sigma_f(sp(c, \mathcal{E})) < \tau \wedge ub(sp(c, \mathcal{E})) \geq \tau\}$
12:        $S := S \cup \{c \in C \mid \sigma_f(sp(c, \mathcal{E})) \geq \tau\}$        ▷ Add any solutions to $S$
13:        $S := S \setminus \{s \in S \mid \exists s' \in S : s \neq s' \wedge s \sqsubseteq s' \wedge sp(s, \mathcal{E}) = sp(s', \mathcal{E})\}$
14:        while $|S| > k$ do
15:            $s \in \arg\min_{s \in S} \sigma_f(sp(s, \mathcal{E}))$        ▷ Remove arbitrary weakest solution
16:            $S := S \setminus \{s\}$
17:        end while
18:        if $|S| = k$ then
19:            $s \in \arg\min_{s \in S} \sigma_f(sp(s, \mathcal{E}))$
20:            $\tau := \sigma_f(sp(s, \mathcal{E}))$                ▷ Update minimum quality threshold
21:        end if
22:        $E := \{e \in E \mid ub(sp(e, \mathcal{E})) \geq \tau\}$       ▷ Filter on minimum quality threshold
23:        while $|E| > E_{max}$ do
24:            $e \in \arg\min_{e \in E} ub(sp(e, \mathcal{E}))$                ▷ Arbitrary weakest candidate
25:            $E := E \setminus \{e\}$
26:        end while
27:        if $(|B| = 0) \wedge (|E| > 0)$ then
28:            $B :=$ REPOPULATEBEAM($E, B_{max}, \mathbf{u}_{OM}, \mathcal{E}$)
29:        end if
30:        if $beam$ then
31:            $E := \emptyset$             ▷ Reject remaining candidates (beam search strategy)
32:        end if
33:        $t := t + n$                  ▷ Increment timer with $n$ time units for this loop
34:    end while
35:    return $S$                                 ▷ Return up to $k$ top solutions
36: end function
```

Gibbs distribution will apportion high probabilities to weak candidates, but as T approaches 0, the distribution will apportion smaller probabilities to weak candidates. Therefore, gradually reducing T over the course of a search initially permits a large number of weak candidates, but over time reduces the probability weak candidates will be accepted over stronger ones into the beam which has the effect of narrowing the search over time. The reduction of the parameter T is user-defined with a *decay* parameter $D \in (0, 1)$, and modifies T from some user-defined initial temperature T_0 every time the REPOPULATEBEAM function is called. As per Definition 3.4.12, Algorithm 9 assumes that the utility function $\mathbf{u}$ ranks preferred candidates with real values which are greater than weak candidates.

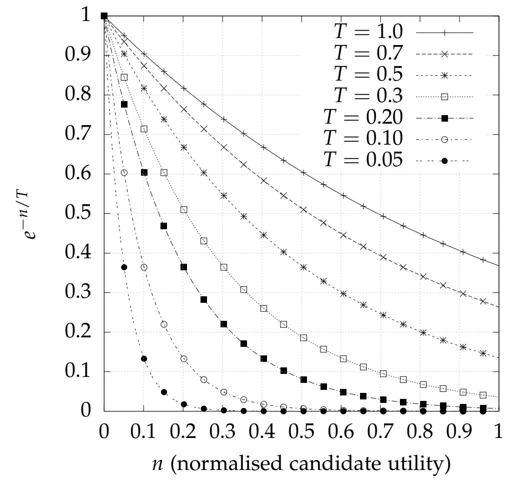


Figure 5.4: A plot of the Gibbs distribution function $e^{\frac{-n}{T}}$ for normalised candidate utility $n \in [0, 1]$ for various temperature values of $T \in [0, 1]$. Note that candidates with normalised utility closer to 0 are considered stronger, with weaker candidates having a normalised utility closer to 1.

Algorithm 8 Best-First Beam Repopulation Algorithm

```
1: $B := \emptyset$
2: **function** REPOPULATEBEAM($E, B_{max}, \mathbf{u}, \mathcal{E}$)
3:     **while** $|B| < B_{max} \land E \neq \emptyset$ **do**
4:         $c \in \arg\max_{c \in E} \mathbf{u}(c, \mathcal{E})$          ▷ Arbitrary best candidate
5:         $E := E \setminus \{c\}$
6:         $B := B \cup \{c\}$
7:     **end while**
8:     **return** $B$
9: **end function**
```

5.1.2.1 Search Parallelisation

Learning Algorithm 7 can be easily parallelised. Any candidate h which is refined to a number of new concepts as $\rho_{\bar{\lambda}}(h)$ on line 10 may each be *evaluated* in parallel. Evaluation requires the computation of the stamp point for each refinement in $h' \in$

Algorithm 9 Stochastic Beam Repopulation Algorithm

```
 1: $B := \emptyset$                                          ▷ Empty beam set
 2: $T := T_0$                                                ▷ Assign initial temperature
 3: function REPOPULATEBEAM($E, B_{max}, \mathbf{u}, \mathcal{E}, D$)
 4:     $U := \emptyset$                                      ▷ Set of candidate/utility pairs
 5:     $(u_{min}, u_{max}) := (-\infty, \infty)$
 6:     for all $c \in E$ do
 7:         $u := \mathbf{u}(c, \mathcal{E})$
 8:         $U := U \cup \{\langle c, u \rangle\}$
 9:         $u_{min} := min\{u_{min}, u\}$
10:         $u_{max} := max\{u_{max}, u\}$
11:     end for
12:     $P := \emptyset$                                      ▷ Set of candidate/probability pairs
13:     $p_{sum} := 0$
14:     for all $\langle c, u \rangle \in U$ do
15:         if $u_{min} = u_{max}$ then
16:             $n := 0$
17:         else
18:             $n := 1 - \frac{u - u_{min}}{u_{max} - u_{min}}$   ▷ Normalise utility into the range $[0, 1]$
19:         end if
20:         $p := e^{-n/T}$                                   ▷ Compute the probability of selecting candidate $c$
21:         $p_{sum} := p_{sum} + p$                          ▷ Accumulate the total probability
22:         $P := P \cup \{\langle c, p \rangle\}$
23:     end for
24:     while $|B| < B_{max} \wedge E \neq \emptyset$ do      ▷ Repopulate the beam set $B$
25:         $\langle c, p \rangle := sample(P, p_{sum})$      ▷ $\langle c, p \rangle \in P$ selected with probability $p/p_{sum}$
26:         $B := B \cup \{c\}$
27:     end while
28:     $T := T \times D$                                     ▷ Decay the temperature for the next invocation
29:     return $B$
30: end function
```

$\rho_{\bar{\lambda}}(h)$, which in turn requires the *coverage* of each h' to be computed over the set of labelled examples $\mathcal{E}$, the process for which will be covered in detail in Section 5.2. As we will see, this is an ideal opportunity to perform processing in parallel as coverage checking may be computationally expensive for knowledge bases containing a large amount of individual and literal data. Alternatively, multiple parallel processing threads may be used to select candidates h from a beam set (line 10) for parallel refinement and evaluation within the main **while** loop, having the advantage of performing refinement *and* evaluation in parallel. This is useful when the refinement operation itself is expensive, as we will explore in Section 6.3.6 of Chapter 6.

5.2 Coverage Checking

In a supervised learning problem with a number of labelled example sets $\mathcal{E}^{\omega}$ for all labels $\omega \in \Omega$, the problem of *coverage checking* seeks to determine, given a concept expression C, which examples e from each labelled set of examples $e \in \mathcal{E}^{\omega}$ lie in the closed-world interpretation $C^{(\mathcal{I},\mathcal{U})}$ for the interpretation $(\mathcal{I},\mathcal{U})$. Each subset of $\mathcal{E}^{\omega}$ covered by $C^{(\mathcal{I},\mathcal{U})}$ as $C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^{\omega}$ provides a way of assessing the quality of C relative to a set of criteria for learning, such as the *utility* of C based on measures like *accuracy* in a classification task.

In any generate-and-test learning scenario over knowledge bases containing a large amount of data, the problem of coverage checking is likely to be the most computationally costly operation. In this section, we will describe a novel method of coverage checking for assessing any concept C given a context-specific interpretation $\mathcal{J}_{\lambda}$ which is approximated by $\mathcal{J}_{\bar{\lambda}}$ as constructed by the instance chase described in Algorithm 4 of Section 4.2.2.

Our method of coverage checking is based on the method referred to as Fast Instance Checking (FIC) as implemented in the DL-Learner system [60]. However, instead of checking if an individual i is an instance of concept C based on the closed-world interpretation $(\mathcal{I},\mathcal{U})$, we make use of the context-specific interpretation $\mathcal{J}_{\bar{\lambda}}$ as maintained in a context graph $\mathcal{G}$ as generated by the instance chase. Algorithm 10 describes the boolean function instanceOf$(i, C, \bar{\lambda})$ which succeeds when i is an instance of C given a set of most applicable contexts $\bar{\lambda}$ and a context-specific interpretation $\mathcal{J}_{\lambda}$.

In contrast with the FIC method, the instance check Algorithm 10 uses local domains Δ_{λ} for each approximated most applicable context $\lambda \in \bar{\lambda}$, which were computed as part of the instance chase of Algorithm 4. In this way, the instance check is able to make use of knowledge captured about individuals chased in each context to limit the amount of checking performed relative to the full closed world interpretation $(\mathcal{I},\mathcal{U})$. For example, consider the instance check of individual i in the role expression $\exists r.D$. Naively, we may compute this check by enumerating all known r-successors of i and testing if at least one is an instance of D. However, if D is complex and i has many r-successors, this may be expensive. Instead, relative to the set of most applicable contexts $\bar{\lambda}$ for $\exists r.D$ and $\bar{\lambda}'$ for D, we can test if i is an instance of $\exists r.D$ by first checking if it is an instance of *all* interpretations $(\exists r.A)^{\mathcal{J}_{\lambda}}$ for each $\lambda \in \bar{\lambda}$ where $\lambda = [\ldots, \exists r.(\circ), A]$. If this test fails for any λ, by Corollary 4.2.21 we know that $i \notin (\exists r.D)^{\mathcal{J}_{\lambda}}$ because $(\exists r.D)^{\mathcal{J}_{\lambda}} \subseteq (\exists r.A)^{\mathcal{J}_{\lambda}}$ holds for all most applicable λ,

Algorithm 10 The boolean **instance check** function which tests if an individual i is an instance of concept C given a context-specific interpretation $\mathcal{J}_\lambda$ and set of most applicable contexts $\bar{\lambda}$.

```
function INSTANCEOF($i, C, \bar{\lambda}$)
    for all $\lambda \in \bar{\lambda}$ do
        if $i \notin \Delta_\lambda$ then
            return false
        end if
    end for
    if $C = A$ or $C = \neg A$ or $C = \{i\}$ then        ▷ If $C$ is a simple concept
        return $i \in C^{(\mathcal{I},\mathcal{U})}$
    else if $C = C_1 \sqcap \ldots \sqcap C_n$ then
        return INSTANCEOFCONJ($i, C_1 \sqcap \ldots \sqcap C_n, \bar{\lambda}$)
    else if $C = C_1 \sqcup \ldots \sqcup C_n$ then
        return INSTANCEOFDISJ($i, C_1 \sqcup \ldots \sqcup C_n, \bar{\lambda}$)
    else if $C = \Diamond r.D$ then
        for all $\lambda \in \bar{\lambda}$ where $\lambda = [\ldots, \Diamond r.(\circ), A]$ do
            if $i \notin (\Diamond r.A)^{\mathcal{J}_\lambda}$ then
                return false
            end if
        end for
        switch $\Diamond$ do
            case $\exists$
                return $\exists \langle i, j \rangle \in r^{\mathcal{J}_{\bar{\lambda}}}$ s.t. INSTANCEOF($j, D, \bar{\lambda}'$)
            case $\forall$
                return $\forall \langle i, j \rangle \in r^{\mathcal{J}_{\bar{\lambda}}}$ : INSTANCEOF($j, D, \bar{\lambda}'$)
            case $\geqslant n$
                return $|\{j \mid \forall j.\langle i, j \rangle \in r^{\mathcal{J}_{\bar{\lambda}}}$ : INSTANCEOF($j, D, \bar{\lambda}'$)$\}| \geq n$
            case $\leqslant n$
                return $|\{j \mid \forall j.\langle i, j \rangle \in r^{\mathcal{J}_{\bar{\lambda}}}$ : INSTANCEOF($j, D, \bar{\lambda}'$)$\}| \leq n$
    end if
end function

function INSTANCEOFCONJ($i, C_1 \sqcap \ldots \sqcap C_n, \bar{\lambda}$)
    return $\bigwedge_{1 \leq j \leq n}$ INSTANCEOF($i, C_j, \bar{\lambda}$)
end function

function INSTANCEOFDISJ($i, C_1 \sqcup \ldots \sqcup C_n, \bar{\lambda}$)
    return $\bigvee_{1 \leq j \leq n}$ INSTANCEOF($i, C_j, \bar{\lambda}$)
end function
```

because by definition we know that $D \sqsubseteq A$. This check can be seen on line 14 and is a precursor to performing more instance checking for r-successors in the potentially complex expression D.

Generally, it is faster to check if an individual i is an instance of a simple concept A or $\neg A$ than it is to check membership in quantified role expressions, as the former simply require a single lookup to see if $i \in A^{(\mathcal{I},\mathcal{U})}$ or $i \in \neg A^{(\mathcal{I},\mathcal{U})}$ which were precomputed prior to executing the learning algorithm. The precedence operator $\preceq$ used in the construction of conjunctive concept expressions ensures that simple concepts appear before role expressions. Therefore, if an individual fails to be an instance of some simple concept A in a conjunction, this will be detected before checking potentially more complex role expressions, permitting the instance check to fail fast where possible.

Given a set of labelled examples $\mathcal{E} = \bigcup_{\forall \omega \in \Omega} \mathcal{E}^{\omega}$ for some $|\Omega| \geq 2$, the cover of any concept C is computed as the set:

$$cover(C, \mathcal{E}) = \{e \in \mathcal{E} \mid instance(e, C, \bar{\lambda})\}$$

The intersection of this set *cover*$(C, \mathcal{E})$ with each labelled set of examples $\mathcal{E}^{\omega}$ for each $\omega \in \Omega$ gives rise to the *stamp point* of C as follows:

$$sp(C, \mathcal{E}) = \langle |cover(C, \mathcal{E}) \cap \mathcal{E}^{\omega_1}|, \ldots, |cover(C, \mathcal{E}) \cap \mathcal{E}^{\omega_n}| \rangle$$

for each $1 \leq i \leq n$ where $n = |\Omega|$. As we saw in Section 5.1.1.2, the stamp point of a concept C relative to labelled examples $\mathcal{E}$ is used to assess the performance of C relative to a measure function σ_f in a learning problem.

When the set of individuals and literals reachable from each example individual in $\mathcal{E}$ in a knowledge base is large, the computation of the cover of any concept may be an expensive operation, as we will discuss in Section 5.2.1.

5.2.1 Computational Complexity of Coverage Checking

The complexity of coverage checking for any concept $C \in \mathcal{L}$ under a closed-world interpretation $(\mathcal{I}, \mathcal{U})$ is different from the complexity of the typical *instance checking* problem for DLs which assume an open-world interpretation $\mathcal{I}$. As discussed in Section 3.2.1.1 of Chapter 3, under an open-world interpretation, instance checking is reducible to satisfiability checking for most DLs, where sound and complete satisfiability checking can be as computationally expensive as N2ExpTime [47].

Under a closed-world interpretation, concepts can be thought of as *queries* over the fixed model $(\mathcal{I},\mathcal{U})$ which can be thought of as a *database*. In this way, the complexity of coverage checking can be analysed as a function of the complexity of the concept C as the query, and the size of the interpretation $(\mathcal{I},\mathcal{U})$ as the database. Central to analysing the complexity of coverage checking is the complexity of the INSTANCEOF function of Algorithm 10 which closely models Definition 3.5.2 of $(\mathcal{I},\mathcal{U})$ in computing whether some individual i is an instance of a concept C with optimisations around context-specific local domains.

We will analyse the complexity of the INSTANCEOF function on a case-by-case basis. Firstly, performing an instance check $i \in C^{(\mathcal{I},\mathcal{U})}$ where C is a simple concept such as any atomic A, negated atomic $\neg A$, or nominal concept $\{i\}$ is an $\mathcal{O}(n)$ operation where $n = |C^{(\mathcal{I},\mathcal{U})}|$, as these concepts will already have their closed-interpretation under $(\mathcal{I},\mathcal{U})$ pre-computed, so the instance check is as complex as set membership.

For concepts which are conjunctions $A_0 \sqcap \ldots \sqcap A_j$ or disjunctions $A_0 \sqcup \ldots \sqcup A_j$ of simple conceptsA_i for $2 \leq i \leq j$, the complexity is at most $\mathcal{O}(j \cdot n)$ where n is the size of the largest interpretation $|A_i^{(\mathcal{I},\mathcal{U})}|$ of any conjunct or disjunct operand A_i.

We now consider the complexity of instance checking for quantified role expressions such as $\Diamond r.(D)$. The complexity of instance checking $i \in (\Diamond r.(D))^{(\mathcal{I},\mathcal{U})}$ is a function of the maximum possible size of the set of all r-successors for any predecessor i which we will denote b, and the worst-case complexity $\mathcal{O}(\psi)$ of performing the instance check against concept D. If D is a simple concept, or a conjunction or disjunction of simple concepts, the complexity of the instance check is then $\mathcal{O}(b \cdot j \cdot n)$ as we potentially check *all* b of i's r-successors in D.

Now assume that D in $\Diamond r.(D)$ is a quantified role expression $\Diamond r.(E)$ where the instance check in E has complexity $\mathcal{O}(\psi)$. The complexity of checking $i \in (\Diamond r.(\Diamond r.(E)))^{(\mathcal{I},\mathcal{U})}$ is $\mathcal{O}((b \cdot (b \cdot \psi)) = \mathcal{O}(b^2 \cdot \psi)$, as at worst, we are required to check that all r-successors of r-successors of i are in E. For further nestings of quantified role expressions, the complexity of instance checking is at least $\mathcal{O}(b^d \cdot \psi)$ where b is the maximum number of any r-successors of any individual, and d is the maximum depth of nested quantified role expressions.

Finally, we consider expressions which may permit simple concepts and nested quantified role expressions along with conjunctions and disjunctions, which represents the full expressivity of concepts which may be generated by refinement operators such as $\rho_{\bar{\lambda}}$. Assume that the maximum number of r-successors for any role r and any individual i is b, and that the maximum number of operands in any conjunct or disjunct is j. For example, consider $\Diamond r.(C_0 \sqcap \ldots \sqcap C_j)$ where any $C_i = \Diamond r.(A)$ for $2 \leq i \leq j$ where A is a simple concept. The complexity of instance checking for such expressions is $\mathcal{O}(b \cdot \sum_j bn) = \mathcal{O}(b^2 \cdot j \cdot n)$.

Now assume each operand C_i it itself a nested role expression which has, as a filler, further conjunctions or disjunctions of nested role expressions where the maximum depth of any nested role expression is d. At the outermost conjunction or disjunction with j operands, there are $b \cdot j$ instance checks for each role expression, and a subsequent $b \cdot j$ for each operand of the conjunction or disjunction in the fillers of each, until we eventually reach simple concepts or conjunctions or disjunctions thereof with instance check complexity n. This results in an overall worst-case complexity of $\mathcal{O}((j \cdot b)^d \cdot n)$ as we check the r-successors of all j operands in conjunctions or disjunctions as role fillers with a maximum depth of d.

In practice, the cost of computing set membership in the pre-computed closed-world interpretation is closer to $\mathcal{O}(1)$ when implemented with hash tables, so the dominating factor in this result is essentially the maximum number of r-successors b for any predecessor individual and role r, the maximum nested role depth d, and the maximum number of operands j for any subexpression which is a conjunction or disjunction. Furthermore, we observe that role depth in concepts is often limited to small values such as less than 10, but this ultimately depends on the structure of the examples in the knowledge base.

At most, this places the complexity of closed-world instance checking over a concept C in the class of ExpTime problems. When compared to instance checking by open-world reasoning, the integration of C into a $\mathcal{SROIQ}$ knowledge base for re-classification to permit instance checking by entailment is a relatively very expensive operation which is a function of the size of the background knowledge as well as the ABox, and is an N2ExpTime problem [47]. In practice, we observe that classification of certain knowledge bases may take minutes whereas closed-world instance checking will often take milliseconds over the same knowledge base, and is therefore clearly preferable for learning by generate-and-test methods. We analyse this behaviour in practice in Chapter 6, Section 6.3.5.

When computing the *coverage* of a concept C relative to a set of example individuals $\mathcal{E}$, we perform the instance check procedure at most $|\mathcal{E}|$ times with the INSTANCEOF function. As $\mathcal{E}$ is of constant size as well as the maximum number of r-successors b for any role, the computational complexity of coverage checking remains the same as that of instance checking, namely $\mathcal{O}((j \cdot b)^d \cdot n)$. Despite the expensive exponential worst-case computational complexity of instance checking, there are several practical optimisations which can significantly improve the performance. Firstly, given that quantified role expressions are the most expensive concepts to check instance membership, it is prudent to perform instance checking in conjunctive and disjunctive expressions against any simple operands first, as it is cheaper to recognise failure (in the case of conjunctions) or success (in the case of disjunctions) against such atomic expressions before checking more expensive role expressions. In the definition of the operator $\rho_{\bar{\lambda}}$, we find that the precedence operator $\preceq$ ensures that atomic operands always appear before quantified role expressions, which supports this optimisation. Secondly, note that the implementation of INSTANCEOF as shown in Algorithm 10 incorporates approximate local domains Δ_λ for all most-applicable domains $\bar{\lambda}$ for any subexpression. By testing if an individual i is not an instance of some approximate local domain Δ_λ, we can be assured that i is also not an instance of any concept C for which $\bar{\lambda}$ was *most appliable*, where $C^{(\mathcal{I},\mathcal{U})} \subseteq \Delta_\lambda$. This approach therefore permits fast-failure on checking potentially expensive role expressions.

Two other optimisations are the caching of concept covers (§5.2.1.1) and fast-failure given minimum bounds on concept performance relative to a convex measure function (§5.2.1.2), which we will now describe.

5.2.1.1 Caching of Concept Covers

In the computation of a stamp point such as $\langle x_0, y_0 \rangle$ for some concept C over a binary labelled set of examples $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, the stamp point $\langle x_1, y_1 \rangle$ of *any* refinement $D \in \rho^*(C)$ will necessarily have $x_1 \leq x_0$ and $y_1 \leq y_0$, as the cover of any refinement D of C will always be a non-strict subset of the cover of C:

$$D \sqsubseteq C \rightarrow cover(D, \mathcal{E}) \subseteq cover(C, \mathcal{E})$$

because if $D \sqsubseteq C$, then by definition of the closed-world interpretation $(\mathcal{I}, \mathcal{U})$, it must be the case that $D^{(\mathcal{I},\mathcal{U})} \subseteq C^{(\mathcal{I},\mathcal{U})}$. Therefore, when computing the stamp point for any concept D, it suffices to begin computation from the cover of its parent concept C where $C \rightsquigarrow_\rho D$. Therefore, we may trade the computational cost of the time spent computing the cover of any concept D over the entire set of examples $\mathcal{E}$ with the space required to maintain the cover of its parent concept, C. While this may increase the space used by a learning algorithm, it may be used to reduce the computation time of coverage checking which is useful when the knowledge base contains a large amount of data, and where candidate hypotheses often cover fewer examples than in $\mathcal{E}$. However, as we observe in Section 6.3.5, if candidates in the search often cover a significant proportion of examples in $\mathcal{E}$, the difference in the performance of coverage testing with caching can be negligible at the cost of increased memory usage.

5.2.1.2 Fast-Failure of Instance Checking with Bounded Convex Measures

Another optimisation to coverage checking which does not require additional space is to leverage the upper bounds on values of a convex measure function σ_f given the cover of a candidate C. As shown in Algorithm 7, any candidate C which does not have an upper bound ub_{σ_f} which exceeds a minimum threshold on quality τ_{min} will be pruned from the search, as it can never lead to a concept with a performance which exceeds τ_{min} for σ_f. As shown in Definition 5.1.3, the upper bound function ub_{σ_f} is defined as the maximum of either $\sigma_f(\langle x, 0\rangle)$ or $\sigma_f(\langle 0, y\rangle)$, over which the threshold τ_{min} is imposed, at least for binary labelled examples. By re-arranging the definition of σ_f in terms of x for $\sigma_f(\langle x, 0\rangle) \geq \tau_{min}$ and for y where $\sigma_f(\langle 0, y\rangle) \geq \tau_{min}$, we obtain two inequalities which impose minimum bounds on the values of x and y, which correspond to the number of labelled examples from each class $x = |C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^+|$ and $y = |C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^-|$, as demonstrated in Example 5.2.1.

Example 5.2.1. *Given a stamp point $\langle x, y\rangle$, the MCC measure $\sigma_{mcc}(\langle x, y\rangle)$ and upper bound $ub_{\sigma_{mcc}}(\langle x, y\rangle)$ as defined on page 120, we re-arrange to obtain the following two inequalities:*

$$x \geq \frac{\tau^2 P(N+P)}{\tau^2 P + N} \qquad\qquad y \geq \frac{\tau^2 N(N+P)}{\tau^2 N + P}$$

These define the minimum number of examples x which a candidate C with stamp point $\langle x, y\rangle$ must cover from $\mathcal{E}^+$ or the minimum number of examples y candidate C must cover from $\mathcal{E}^-$ for the value of measure σ_{mcc} to meet or exceed τ.

Generally, for stamp points $\langle c_1, \ldots, c_n\rangle$ in labelled learning problems where $|\Omega| = n$, each variable c_i where $1 \leq i \leq n$ corresponds to the number of examples in the cover of some concept which are labelled with $\omega_i \in \Omega$. As shown in Example 5.2.1, we computed the minimum bounds on each variable c_i which satisfy an inequality $\sigma_f(\langle c_1, \ldots, c_n\rangle) \geq \tau$ where f was MCC by re-arranging for each c_i to produce $c_i \geq \Phi(\sigma_f, i, \tau)$ where $\Phi(\sigma_f, i, \tau)$ denotes the right hand side of the re-arrangement of the inequality for c_i.

For a candidate C with stamp point $\langle c_1, \ldots, c_n\rangle$ to be pruned from a search based on insufficient upper bounds on the performance of σ_f, it must be the case that $c_i < \Phi(\sigma_f, i, \tau)$ for *all* $1 \leq i \leq n$, as all downward refinements D of C where $D \in \rho^*(C)$ with stamp point $\langle c'_1, \ldots, c'_n \rangle$ will never satisfy $\sigma_f(\langle c'_1, \ldots, c'_n \rangle) \geq \tau$.

Algorithm 11 computes a *coverage* for a concept C as a tuple $\langle I_1, \ldots, I_n \rangle$ where each I_i for $1 \leq i \leq n$ is the set of examples which are instances of C labelled $\omega_i \in \Omega$. Such tuples can be used to compute the stamp point of C relative to a set of labelled examples $\mathcal{E}$ as $\langle |I_1|, \ldots, |I_n| \rangle$. However, Algorithm 11 is designed such that if it can determine over the course of execution that, for each I_i where $1 \leq i \leq n$ the inequality $|I_i| \geq \Phi(\sigma_f, i, \tau)$ cannot be satisfied, each I_i may contain fewer examples than is actually in the cover of C as it fails fast on the expectation that C will be pruned from the search as its stamp point will not satisfy the minimum bound τ on the measure σ_f. Furthermore, Algorithm 11 ensures that, if any I_i does satisfy its related inequality $|I_i| \geq \Phi(\sigma_f, i, \tau)$, that each I_i in the computed tuple $\langle I_1, \ldots, I_n \rangle$ will contain the exact number of examples which are instances of C with label ω_i.

5.3 Search Efficiency

Methods of supervised learning such as the top-k search of Algorithm 7 are designed to be efficient by pruning the search space where possible. However, the performance of such search methods ultimately depends on the behaviour of the refinement operator $\rho_{\bar{\lambda}}$ as presented in Section 4.3 of Chapter 4 which is used to structure and traverse the space of concepts towards solutions. Recall that the operator $\rho_{\bar{\lambda}}$ has the properties of being *redundant* and *improper*, which are both detrimental to the performance of a search algorithm. The property of redundancy means that the search will potentially re-visit the same concept more than once which is clearly wasteful of resources. The property of improperness describes how the operator, when refining any concept C, may produce an equivalent concept D where $D \in \rho_{\bar{\lambda}}(C)$ for which $C \equiv D$. Concept equivalence here is defined in terms of the closed-world interpretation where $C \equiv D$ means $C^{(\mathcal{I},\mathcal{U})} = D^{(\mathcal{I},\mathcal{U})}$, as the refinement operator ensures that the concept expression D is not structurally identical to C. Improperness can be detrimental to the performance of a search algorithm because the measures used to assess C and D are based primarily on their coverage that is essentially their interpretation with respect to a set of examples $\mathcal{E}$. If the operator generates multiple improper refinements $D_1, \ldots, D_n \in \rho_{\bar{\lambda}}(C)$ where $\forall_{1 \leq i \leq n}\ D_i^{(\mathcal{I},\mathcal{U})} = C^{(\mathcal{I},\mathcal{U})}$, methods such as the search Algorithm 7 which select the concepts for further refinement us-

Algorithm 11 The **coverage** algorithm which computes the vector $\langle I_1, \ldots, I_n \rangle$ as the basis for a stamp point of a concept C given labelled example sets $\langle \mathcal{E}^{\omega_1}, \ldots, \mathcal{E}^{\omega_n} \rangle$ for $n \geq 2$ and where each $I_i \subseteq C^{(\mathcal{I},\mathcal{U})} \cap \mathcal{E}^{\omega_i}$ for $1 \leq i \leq n$. The vector $\langle |I_1|, \ldots, |I_n| \rangle$ is the *stamp point* of concept C. The boolean variable $q = true$ indicates that the auxiliary function coverCheckAux should fail fast when constructing sets I_i, or whether it should compute the full sets I_i for each $1 \leq i \leq n$ when $q = false$.

```
function coverCheck($C, \bar{\lambda}, \tau, \langle \mathcal{E}^{\omega_1}, \ldots, \mathcal{E}^{\omega_n} \rangle$)
    $P := \langle (I_1, O_1, \mathcal{E}^{\omega_1}), \ldots, (I_n, O_n, \mathcal{E}^{\omega_n}) \rangle$ where $\forall_{1 \leq i \leq n} : I_i = \emptyset$ and $O_i = \emptyset$
    $q := true$                                        ▷ Initially, fail fast if possible
    return coverCheckAux($C, \bar{\lambda}, \tau, q, P$)
end function

function coverCheckAux($C, \bar{\lambda}, \tau, q, \langle (I_1, O_1, \mathcal{E}^{\omega_1}), \ldots, (I_n, O_n, \mathcal{E}^{\omega_n}) \rangle$)
    $\bar{b} := \langle b_1, \ldots, b_n \rangle$ where $\forall_{1 \leq i \leq n} : b_i = true$
    for all $\omega_i \in \Omega$ where $1 \leq i \leq n$ do
        for all $e \in \mathcal{E}^{\omega_i} \setminus (I_i \cup O_i)$ do        ▷ For all remaining examples to test
            if instanceOf($e, C, \bar{\lambda}$) then
                $I_i := I_i \cup \{e\}$                   ▷ Example $e$ lies *inside* the cover of $C$
            else
                $O_i := O_i \cup \{e\}$                   ▷ Example $e$ lies *outside* the cover of $C$
            end if
            if $q \wedge (|I_i| + (|\mathcal{E}^{\omega_i}| - |I_i| - |O_i|) < \Phi(\sigma_f, i, \tau))$ then
                $b_i := false$
                break    ▷ Fast fail as $C$ will not cover enough examples labelled $\omega_i$
            end if
        end for
    end for
    if $q \wedge \exists b_i \in \bar{b}$ s.t. $b_i = true$ then
        $q := true$                                   ▷ Complete the evaluation of each $I_i$
        return coverCheckAux($C, \bar{\lambda}, \tau, q, P$)
    end if
    return $\langle I_1, \ldots, I_n \rangle$
end function
```

ing a heuristic function that ranks candidates based on their cover cannot effectively select any concept D_i as being better or worse than any other equivalent concept, and the search may then become unguided. This is primarily a concern when the search method is memory-bounded, as maintaining a limited sized set of candidates containing many equivalent concepts will result in the search behaving in a largely unguided manner where it may not be able to select an appropriate trajectory for traversing the search space towards solutions using the heuristic. Therefore, it is prudent to limit improper refinements where possible to permit a search algorithm to apply any heuristic functions it has to aid in directing the search towards spaces which might contain solutions. In the next two sections, we will discuss how redundancy (§5.3.1) and improperness (§5.3.2) may be identified and mitigated to improve the performance of search methods like Algorithm 7 which rely on operators like $\rho_{\bar{\lambda}}$ which have such undesirable properties.

5.3.1 Redundancy in Refinement

Redundancy in the refinement operator $\rho_{\bar{\lambda}}$ is characterised by the potential for the operator to refine two or more *different* concepts, say, C and D where $C \neq D$, to the *same* concept, E where $E \in \rho^*_{\bar{\lambda}}(C)$ and $E \in \rho^*_{\bar{\lambda}}(D)$. Redundancy therefore describes the situation where multiple different refinement chains which reach E from different parts of the search space may occur. For example, consider the following two refinement chains:

$$A \rightsquigarrow B \rightsquigarrow B \sqcap C$$
$$A \rightsquigarrow C \rightsquigarrow C \sqcap B$$

where $B \sqcap C \equiv C \sqcap B$ as conjunction and disjunction are commutative. In terms of conjunctions, the operator $\rho_{\bar{\lambda}}$ is defined in such a way which prevents redundant refinement chains such as these because it uses a precedence operator $\preceq$ to impose an order on any concept as the operand of a conjunction where only one of $B \sqcap C$ or $C \sqcap B$ would be permitted, such as the latter when $C \preceq B$, and recognises when syntactically equivalent concepts such as $B \sqcap B$ may occur so as to exclude these. Similarly, the operands of disjunctions are constructed so as to ensure that they conform to precedence rules according to $\preceq$, except that disjuncts may be repeated, as in the following refinement chain:

$$A \rightsquigarrow A \sqcup A \rightsquigarrow (A \sqcap \exists r.B) \sqcup A \rightsquigarrow (A \sqcap \exists r.B) \sqcup (A \sqcap \forall s.C)$$

Here, repeated concepts such as A in $A \sqcup A$ are necessary as intermediate steps to reach disjunctive expressions with different operands as shown. By applying the precedence operator in all refinements, the operator $\rho_{\bar{\lambda}}$ is designed to minimise redundancy which may otherwise occur in the construction of conjunctions and disjunctions. However, as we saw in Example 4.3.3, the downward operator $\rho_{\bar{\lambda}}$ may still generate redundant refinement chains depending on the axioms in the TBox.

In a learning method such as Algorithm 7, the refinements of any concept C as $\rho_{\bar{\lambda}}(C)$ is maintained as a set which excludes duplicate concepts under syntactic equality $=$, therefore any redundant refinements in one application of $\rho_{\bar{\lambda}}$ will be eliminated. Similarly, the collection of concepts maintained in a beam or expansion set in a search may also eliminate syntactic duplicates. However, this does not preclude the possibility of the search selecting a candidate C for refinement which generates some D, which is evaluated and either pruned or refined to new concepts. Then, if some other candidate C' is selected which refines to the same concept D later, it will be re-evaluated if it does not appear in the size-limited beam or expansion sets.

One strategy for managing redundancy during the entire execution of a learning method such as Algorithm 7 is to maintain a set of all previously assessed candidate expressions, even if they no longer form part of the current beam or expansion set. In this way, redundancy can be effectively managed by recognising if a syntactically equivalent concept has been evaluated before at any time during execution. However, if the search space of concepts is very large and the algorithm searches a large portion of the space, memory limitations may restrict how many concepts can be maintained for this purpose. If the number of potentially redundant refinement chains is relatively small when compared to the size of the space of all concepts searched by the operator, redundancy may not significantly affect the overall performance of a learning algorithm. In this case, we may permit the re-evaluation of any concept without a noticeable impact on performance. As we will see in Section 6.3.3 of Chapter 6, concepts reachable by redundant refinement chains appear in only a small percentage of the time spent executing Algorithm 7 for the several problems we analyse, leading us to believe that the degree to which the operator $\rho_{\bar{\lambda}}$ generates redundant refinement chains is generally negligible. Therefore, we propose that a strategy that maintains a record of assessed candidate expressions need not be employed when using $\rho_{\bar{\lambda}}$ as the refinement strategy.

5.3.2 Improperness in Refinement

In the last chapter, we described how the refinement operator $\rho_{\bar{\lambda}}$ is *improper*. Improperness of refinement can adversely affect the performance of a refinement-based search such as Algorithm 7 by introducing many equivalent concepts into a limited size beam or expansion set, which we denote in combination as the search *fronter*. If this occurs, the heuristic utility function (Definition 3.4.12) used by the algorithm to select the best candidates for further refinement may not be able to select between the best concepts if there are many equivalent candidates, and the search may become *unguided*. In this case, the search is not able to control the trajectory of the search into the space of concepts toward solutions.

Another complication which can arise from the situation where many improper refinements populate the fixed-sized frontier is as follows. Consider a frontier of fixed-sized n which currently contains candidates $D_1, \ldots, D_i, \ldots, D_n$ where $D_1, \ldots, D_{i-1}$ each have utility greater than u, and where $D_i, \ldots, D_n$ each have utility less than u. Also consider the case where solutions exist only in refinements of $D_i, \ldots, D_n$. As search algorithms like Algorithm 7 are likely to refine the candidates $D_1, \ldots, D_{i-1}$ first as they have greater utility, the fixed-sized frontier may be populated with their refinements to the exclusion of weaker candidates $D_i, \ldots, D_n$ and their refinements. If refinements of the initially stronger candidates $D_1, \ldots, D_{i-1}$ do not contain any solutions whereas those weaker $D_i, \ldots, D_n$ did, the search will not locate any solutions. This situation is illustrated in Figure 5.5 and is a well-known shortfall of the local-search strategy such as that implemented by Algorithm 7 with a fixed beam and expansion set size.

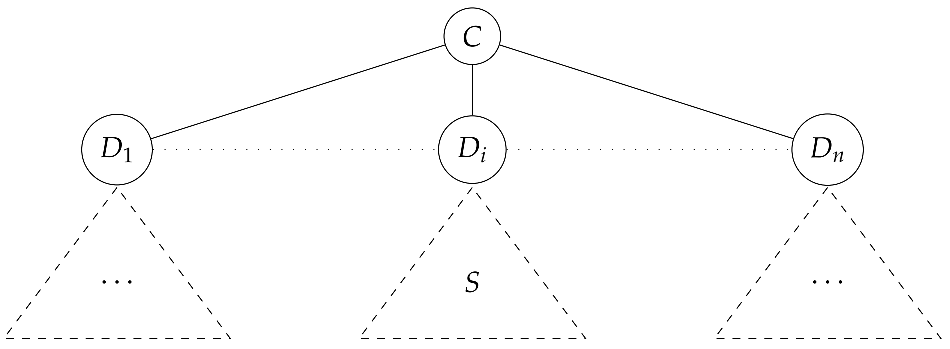


Figure 5.5: Concept expression C refined to $\rho(C) = \{D_1, \ldots, D_i, \ldots, D_n\}$ for $1 \leq i \leq n$ showing solution S where $S \in \rho^*(D_i)$. If concepts $D_i, \ldots, D_n$ are pruned from a memory-bounded search because concepts $D_1, \ldots, D_{i-1}$ are preferred according to a utility function, a learning method such as Algorithm 7 will not find solution S.

The situation illustrated by Figure 5.5 can occur when refining concepts involving disjunction, such as $A \rightsquigarrow A \sqcup A$. As we saw in the previous section, such improper downward refinement steps may be necessary if they act as an intermediate bridge to reach more specific non-equivalent disjunctive concepts which are solutions. Clearly, the utility of such improper steps remains unchanged, however other refinements such as $A \rightsquigarrow B$ or $A \rightsquigarrow A \sqcap C$ may produce concepts of higher utility, but where further refinements may not lead to solutions.

One approach to addressing improperness in refinement which has been discussed in the analysis of ρ_B [58] and which also applies to $\rho_{\bar{\lambda}}$ is to simply permit the repeated application of an improper downward refinement operator ρ under the assumption that proper refinements of predecessor concepts will eventually be reached. This strategy suggests that if a downward refinement operator ρ is improper, we may perform a finite number of refinements of some concept expression $C_0 \rightsquigarrow_\rho C_1 \rightsquigarrow_\rho \ldots \rightsquigarrow_\rho C_n$ where $C_0 \equiv C_i$ for $1 \leq i < n$ until we reach a proper refinement where $C_n \sqsubset C_0$.

The difficulty with this approach is that the number of improper refinements generated by simply permitting the repeated application of the operator ρ may outnumber any proper refinements, and end up dominating a limited-size frontier such that learning becomes unguided, or alternatively, such improper refinements may be pruned from a search in preference for higher-performing proper refinements, such as illustrated in Figure 5.5.

Without modifying the behaviour of the refinement operator itself, this problem could be mitigated by increasing the size of the frontier in a memory-bounded search to accommodate for improper refinements until they can be expanded to proper ones, at which time the heuristic utility function can again determine in which direction to drive the search. However, when the concept space is vast, this is not a practical solution. Instead, we aim to analyse the behaviour of the refinement operator relative to certain concept expressions to identify when improper refinement steps can be recognised. Once we identify such improper refinement steps, we will describe how they can be limited in a way which does not affect the ability of the search to reach solutions.

For example, consider the case where subexpression S_1 is being refined in the concept $T_1 \sqcup S_1$ where $S_1 \sqsubseteq T_1$. In this case, downward refinement may permit the following refinement chain:

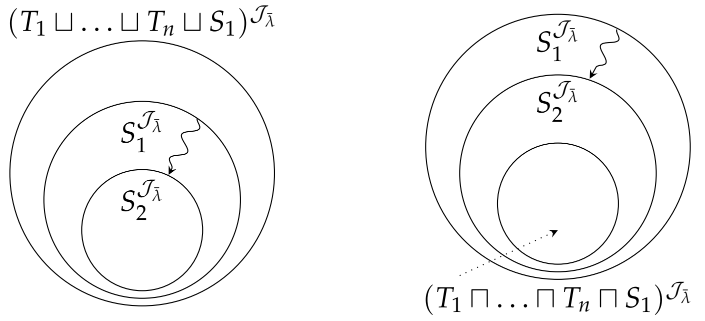
$$T_1 \sqcup S_1 \rightsquigarrow_\rho T_1 \sqcup S_2 \rightsquigarrow_\rho \ldots \rightsquigarrow_\rho T_1 \sqcup S_n \rightsquigarrow_\rho T_1 \sqcup \bot$$

Ultimately, refinement of the subexpressions $S_1, \ldots, S_n$ will never alter the closed-world interpretation of the disjunction $T_1 \sqcup S_i$ for $1 \leq i \leq n$, which is always equivalent to T_1. Note that the space of concepts between S_1 and $\bot$ may be vast, so permitting the refinement operator ρ to produce such refinements is potentially detrimental to a memory-bounded search algorithm. We describe such refinements as being *ineffectual*, as each improper refinement results in another improper refinement, and where the quality of each cannot be assessed by the learning algorithm.

Definition 5.3.1. ***(Ineffectual Downward Refinement)*** *We denote any improper downward refinement step $C \rightsquigarrow D$ where $C \equiv D$ as being* ***ineffectual*** *if the step modifies S_1 where $S_1 \in subex(C)$ in the context(s) denoted by $\bar{\lambda}$ to S_2 where $S_2 \in subex(D)$, where each subexpression was an operand of a conjunction or disjunction T as follows:*

$$\begin{array}{ll} T_1 \sqcup \ldots \sqcup T_n \sqcup S_1 \rightsquigarrow T_1 \sqcup \ldots \sqcup T_n \sqcup S_2 & \textit{where } S_2 \sqsubseteq S_1 \sqsubseteq T_1 \sqcup \ldots \sqcup T_n\textit{, or} \\ T_1 \sqcap \ldots \sqcap T_n \sqcap S_1 \rightsquigarrow T_1 \sqcap \ldots \sqcap T_n \sqcap S_2 & \textit{where } S_1 \sqsupseteq S_2 \sqsupseteq T_1 \sqcap \ldots \sqcap T_n \end{array}$$

Informally, such single-step refinements are denoted as being ineffectual as they do not produce a proper downward specialisation and were unguided in the sense they cannot be assessed based on their cover. These two cases are illustrated in Figure 5.6.



$(T_1 \sqcup \ldots \sqcup T_n \sqcup S_1)^{\mathcal{J}_{\bar{\lambda}}}$
$S_1^{\mathcal{J}_{\bar{\lambda}}}$
$S_2^{\mathcal{J}_{\bar{\lambda}}}$
$S_1^{\mathcal{J}_{\bar{\lambda}}}$
$S_2^{\mathcal{J}_{\bar{\lambda}}}$
$(T_1 \sqcap \ldots \sqcap T_n \sqcap S_1)^{\mathcal{J}_{\bar{\lambda}}}$

Figure 5.6: Illustration of the set-based interpretation of various concepts along with an *ineffectual* refinement step, $S_1 \rightsquigarrow S_2$, as per Definition 5.3.1. In each case, the refinement step $S_1 \rightsquigarrow S_2$ results in an improperly refined concept relative to a closed-world interpretation $\mathcal{J}_{\bar{\lambda}}$.

Generally, we aim to reduce the number of unguided refinements performed by a learning algorithm to improve its chances of making informed decisions about which concepts to maintain in the search frontier. Our strategy to limit the number of ineffectual refinements is to perform what we call *subexpression suspension*. Basically, if it can be recognised that any operand T_i of a disjunction $T_1 \sqcup \ldots \sqcup T_n$ for $1 \leq i \leq n$ is subsumed by $T_1 \sqcup T_{i-1} \sqcup T_{i+1} \sqcup T_n$, then we *suspend* the refinement of T_i and instead permit refinement of all other subsuming operands until T_i is no longer subsumed. Similarly, if it can be recognised that any operand T_i of a conjunction $T_1 \sqcap \ldots \sqcap T_n$ subsumes $T_1 \sqcap T_{i-1} \sqcap T_{i+1} \sqcap T_n$, we may *suspend* the refinement of all other operands until T_i is refined to an expression which no longer subsumes the conjunct of the other operands. This method attempts to force the operands of disjunctions to cover overlapping or disjoint sets of data, and the operands of conjunctions to cover overlapping sets of data. In this way, refinements of any operand of a disjunction or conjunction which are subexpressions of some candidate expression C is expected to generate a proper refinement of C in fewer refinement steps.

Given two concept expressions T_i and T_j which are operands of a disjunction or conjunction, testing whether $T_i \sqsubseteq T_j$ either requires checking if $\mathcal{K} \models T_i \sqsubseteq T_j$ via logical reasoning under the open-world interpretation $\mathcal{I}$ or directly via the closed-world interpretation $(\mathcal{I},\mathcal{U})$ as $T_i^{(\mathcal{I},\mathcal{U})} \subseteq T_j^{(\mathcal{I},\mathcal{U})}$. Under $\mathcal{I}$, the complexity of checking a subsumption relationship $T_i \sqsubseteq T_j$ between two concept expressions previously unseen by the knowledge base requires re-classification of the TBox, which may be computationally expensive for highly expressive DLs as discussed in Section 3.2.1.1 of Chapter 3. Instead, we can leverage the closed-world interpretation $(\mathcal{I},\mathcal{U})$ which was pre-computed prior to learning in order to check $T_i \sqsubseteq T_j$ via the coverage of each T_i and T_j. For every candidate C under consideration which contains a number of subexpressions which are conjunctions or disjunctions, a naive method for computing the subsumption relationship between any two operands is to independently compute the cover of each based on the set of individuals in the knowledge base and then test if $T_i^{(\mathcal{I},\mathcal{U})} \subseteq T_j^{(\mathcal{I},\mathcal{U})}$ or $T_j^{(\mathcal{I},\mathcal{U})} \subseteq T_i^{(\mathcal{I},\mathcal{U})}$. However, this is potentially computationally expensive if the set of individuals in $\Delta^{(\mathcal{I},\mathcal{U})}$ is large. Instead, we propose to modify the INSTANCEOF procedure of Algorithm 10 which tests membership of individuals in conjunctions or disjunctions with the INSTANCEOFCONJ and INSTANCEOFDISJ auxiliary functions.

Consider a candidate concept C in a learning method such as Algorithm 7. Utility functions such as $\mathbf{u}_{OM}(C, \mathcal{E})$ which are used to assess the performance of C do so relative to the cover of C over the set of labelled examples $\mathcal{E}$ with a procedure such as Algorithm 11, which in turn uses Algorithm 10 to check if example instances $e \in \mathcal{E}$

belong in the cover of C as $e \in C^{(\mathcal{I},\mathcal{U})}$. If the candidate C contained any conjunction or disjunction as a subexpression at the exact context λ, then during the coverage check process, we aim to record for each operand T of the conjunction or disjunction in context λ the set of individuals which were assessed as being an instance of T as the *partial cover* denoted $I_{(T,\lambda)}$ relative to a context-specific closed-world interpretation $\mathcal{J}_\lambda$, where $I_{(T,\lambda)} \subseteq T^{\mathcal{J}_\lambda}$. Note that the instance check Algorithm 10 relies on two auxiliary functions for testing instance membership in conjunctions and disjunctions with the functions INSTANCEOFCONJ and INSTANCEOFDISJ. Here, we will modify these functions to record the sets $I_{(T_i,\lambda)}$ for every conjunct or disjunct operand T_i over the course of an execution of the COVERCHECK function over C. Once complete, subsumption checking between the operands T_i, T_j of any conjunction or disjunction at context λ in C can then be inferred by testing whether $T_i \sqsubseteq T_j$ if $I_{(T_i,\lambda)} \subseteq I_{(T_j,\lambda)}$. Note that this method is approximate, as the subsumption test relies on the subset S of individuals selected for instance checks against conjunctions or disjunctions which will be a subset of all individuals in the context-specific interpretation of any operand T_i as $S \subseteq T_i^{\mathcal{J}_\lambda}$. Nevertheless, if a subsumption relationship exists between any two operands in a conjunction or disjunction, this will still be apparent relative to the sets $I_{(T_i,\lambda)}$ which are a subset of S. Algorithm 12 presents modifications to the instance check functions to permit the construction of sets $I_{(T_i,\lambda)}$.

Once a coverage check is computed for some concept C via the COVERCHECK function of Algorithm 11 which makes use of the INSTANCEOF function and auxiliary functions of Algorithm 12, a number of sets $I_{(T_i,\lambda)}$ are generated for each conjunct or disjunct operand with subexpression context λ in C. Once computed, these sets may be analysed to infer approximate subsumption relationships between operands of each conjunction or disjunction for the purposes of suspension. Note that we only intend to suspect quantified role expressions $\Diamond r.(D)$, as these are the only kind of expressions which can be refined further. To this end, we only compute subsumption for operands which are role expressions as follows. Given an exact subexpression context λ which identifies a disjunction $T_1 \sqcup \ldots \sqcup T_n$ in a concept C, the set of *subsumed* role expression operands $S_{\lambda,\sqcup}$ are computed as follows:

$$S_{\lambda,\sqcup} = \{T_i \mid \forall_{1 \leq i \leq n} T_i : T_i = \Diamond r.D \wedge I_{(T_i,\lambda)} \subseteq \bigcup_{\forall j \neq i} : I_{(T_j,\lambda)}\}$$

for any role name r, concept D and quantifier $\Diamond$. Similarly, when λ identifies a conjunction $T_1 \sqcap \ldots \sqcap T_n$ in a concept C, the set of *subsuming* role expression operands

Algorithm 12 Alternate implementations of the instance check functions for conjunction and disjunctions to support *subexpression suspension* for limiting ineffectual refinements.

```
function instanceOfConj($a, C_1 \sqcap \ldots \sqcap C_n, \bar{\lambda}$)
    return instanceOfCheck($a, \sqcap, \{C_i \mid 1 \leq i \leq n\}, \bar{\lambda}$)
end function

function instanceOfDisj($i, C_1 \sqcup \ldots \sqcup C_n, \bar{\lambda}$)
    return instanceOfCheck($a, \sqcup, \{C_i \mid 1 \leq i \leq n\}, \bar{\lambda}$)
end function

function instanceOfCheck($a, con, \{C_1, \ldots, C_n\}, \bar{\lambda}$)
    $I := \emptyset$
    for all $C_i \in \{C_1, \ldots, C_n\}$ do
        if instanceOf($a, C_i, \bar{\lambda}$) then
            $I_{(C_i,\lambda)} := I_{(C_i,\lambda)} \cup \{a\}$        ▷ Where $\lambda$ is the exact subex context of $C_i$
            $I := I \cup \{C_i\}$
        end if
    end for
    switch $con$ do
        case $\sqcap$
            return $|I| = n$                  ▷ $a$ an instance of all $n$ conjunct operands
        case $\sqcup$
            return $|I| > 0$          ▷ $a$ an instance of at least one disjunct operand
end function
```

$S_{\sqcap}$ are computed as follows:

$$S_{\lambda,\sqcap} = \{T_i \mid \forall_{1\leq i\leq n} T_i : T_i = \Diamond r.D \wedge I_{(T_i,\lambda)} \supseteq \bigcup_{\forall j\neq i} : I_{(T_j,\lambda)}\}$$

Once computed for each exact subexpression context λ, the sets $S_{\lambda,\sqcup}$ or $S_{\lambda,\sqcap}$ contain the operands which may be temporarily suspended until the conditions of being subsumed no longer hold after future refinements of all other non-suspended operands. If either set $S_{\lambda,\sqcup}$ or $S_{\lambda,\sqcap}$ contains *all* operands of their respective disjunction or conjunction subexpressions, then one operand is arbitrarily removed to permit the refinement operator to specialise the expression in some way, unless no refinements were generated on the operand in a previous application of $\rho_{\bar{\lambda}}$. This requires us to be able to *label* any subexpression T with context λ of a candidate concept C to indicate whether:

1. $\rho_{\bar{\lambda}}(T) = \emptyset$: No refinements are possible, so we label T at λ in C as $\underline{T}$.
2. T is *suspended* in C at λ because it appeared in a set $S_{\lambda,\sqcap}$ or $S_{\lambda,\sqcap}$: No refinements should be performed, so we label T at λ in C as $\widetilde{T}$.

We then modify the behaviour of the refinement operator $\rho_{\bar{\lambda}}$ such that it inspects the label of any subexpression under consideration for refinement such that:

1. $\rho_{\bar{\lambda}}(\underline{T}) = \emptyset$: No refinements are attempted because a previous attempt produced no refinements.
2. $\rho_{\bar{\lambda}}(\widetilde{T}) = \emptyset$: No refinements are attempted because T is temporarily *suspended* in C.

Before the coverage check is performed for any candidate C which computes the subsumption relationships between operands of conjunctive and disjunctive subexpressions of C, all suspension labels $\widetilde{T}$ are cleared so that new labels may be applied after the previous refinement.

While the suspension method we have described will not eliminate improper refinements, it modifies the behaviour of $\rho_{\bar{\lambda}}$ to reduce ineffectual refinements while not preventing the search from being able to reach any concept which would otherwise be reachable with $\rho_{\bar{\lambda}}$ without using the suspension method. As discussed in Section 6.3.4 of Chapter 6, we have tested the use of the suspension method across several learning problems where we have found that the method significantly reduces the number of improper refinements overall while still permitting the search to locate high quality results as expected. Furthermore, subjective analysis of the concepts produced as solutions to the problems tested with the suspension method appear to contain fewer terms without a reduction in quality compared to those produced without suspension, suggesting that the method produces more readable and compact concepts overall.

5.4 Summary

In this chapter, we have explored the topic of supervised learning in DL knowledge bases, particularly around the problems of classification and subgroup discovery (§5.1). We analysed the basic beam search strategy and described the use of heuristics which guide beam search methods, along with an analysis of the properties of various measure functions as used by heuristic functions (§5.1.1). We then presented our novel search algorithm which leverages existing work on the properties of measure functions in an improved memory-bounded beam search (§5.1.2). We then presented our novel method of efficient coverage checking which utilises the context graph as introduced in the previous chapter (§5.2). Finally, we discussed various inefficiencies of our search method based on limitations of our refinement operator $\rho_{\bar{\lambda}}$ and presented novel methods to mitigate these problems (§5.3). In the next chapter, we describe our implementation OWL-Miner and evaluate its performance over several well-known benchmark problems.

Chapter 6

Implementation and Evaluation

In this chapter, we describe the OWL-Miner system which is our software implementation of the various methods presented in this thesis (§6.1). We then present an evaluation of the OWL-Miner system against particular well-known benchmark problems using either classification or subgroup discovery (§6.2). We then analyse the performance of certain novel methods which the OWL-Miner system implements (§6.3), including the construction of the context graph and its use in refinement (§6.3.1, 6.3.2) to the performance of the instance check and coverage computation procedures (§6.3.5). We then remark on the effect of parallelisation on the search (§6.3.6). Overall, we find that the OWL-Miner system achieves strong results in terms of the quality of solutions found for classification and subgroup discovery and generally outperforms similar systems such as DL-Learner and various ILP systems which are designed to solve similar problems.

6.1 Implementation: OWL-Miner

We have developed a new DL-learning system called OWL-Miner for classification and subgroup discovery which implements the methods introduced in this thesis. The OWL-Miner system has been developed in Java 8 in an extensible way to support new refinement operators, convex measure functions and heuristic utility functions. The OWL-Miner system is a fresh implementation of a DL-learner and is distinct from the open-source DL-Learner software[1]. This is necessary as the OWL-Miner implementation relies on methods which are significantly different from DL-Learner. The main differences are that OWL-Miner constructs a *context graph* (see Definition 4.2.12) with the *instance chase* of Algorithm 4, and implements refinement operators such as $\rho_{\bar{\lambda}}$ (see Definition 4.5.1) to traverse the context graph and to con-

[1] DL-Learner is available from: http://dl-learner.org/

struct *most applicable contexts* $\bar{\lambda}$ for every subexpression of a concept being refined in order to assess suitable refinement options. The coverage computation by Algorithm 11 and instance check by Algorithm 10 are also different in that they rely on analyses of convex measure functions for fast-failure, and are also used to determine hypothesis subexpression suspension options (§5.3.2) for limiting the production and evaluation of poor candidate hypotheses. Furthermore, the generalised top-k concept search method of Algorithm 7 is novel and relies on a large amount of thread-safe code to support parallel execution. While these methods are general in nature and could be integrated into the open-source DL-Learner project, we have left this task to future work. The OWL-Miner system relies on several third-party libraries, including the following:

- Pellet [92] or HermiT [91] for open-world DL reasoning;
- The OWL-API library [41];
- JGraphT (http://jgrapht.org) for creating, manipulating and querying graph structures;
- RabbitMQ (https://www.rabbitmq.com/) for accepting learning tasks and publishing results for integration with the X-plorer system (§7.4);
- Various Apache Commons libraries, including:
  - Math (http://commons.apache.org/proper/commons-math/);
  - Lang (http://commons.apache.org/proper/commons-lang/);
  - CLI http://commons.apache.org/proper/commons-cli/);
  - Collections http://commons.apache.org/proper/commons-collections/).

OWL-Miner is designed to support two modes of operation: *batch*, and *online*. Batch mode requires a single configuration file that references an OWL file containing an ontology and individual examples, and outputs an OWL file containing hypothesis classes which are solutions to a particular learning problem along with various statistics for each hypothesis relative to the chosen measure function. Online mode is similar to batch mode in that requires a single configuration file that references an OWL file containing an ontology and individual examples, but then listens for requests to perform certain learning tasks on a message channel. Once received, OWL-Miner will then process a learning task specification and will output the result as a JSON file on the return channel for consumption by the requesting agent. In either mode, the flow of major steps in these processes is illustrated in Figure 6.1.

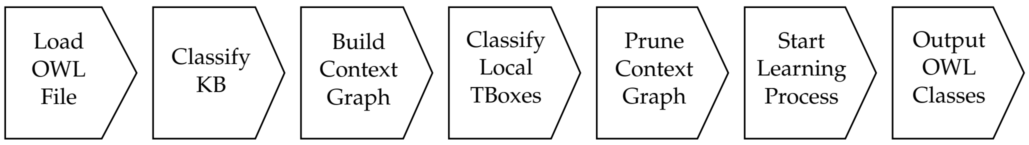


Figure 6.1: The overall OWL-Miner system process flow.

The OWL-Miner process flow of as depicted in Figure 6.1 can be described as follows:

- **Load OWL File.** An OWL file which contains an ontology as a set of classes, properties, class inclusion axioms and data assertions is loaded. This file describes the DL knowledge base $\mathcal{K}$ and contains the axioms of the TBox and ABox which are used in the system.
- **Classify KB.** Initially, a DL-reasoner such as Pellet [92] is used to determine all known individuals for each named concept (atomic OWL class) and role (OWL object and datatype property) using open-world reasoning. From this set of all entailed assertions, the fixed closed-world IC interpretation (Definition 3.5.2) of every atomic concept and role is computed, and the weak UNA is adopted (Definition 3.5.3). This step produces the fixed closed-world model over which DL-learning will proceed.
- **Build Context Graph.** Given a set of example individuals labelled within the input ontology, the context graph is constructed (§4.2.2, Definition 4.2.12) with the instance chase by Algorithm 10. The context graph captures the various ways the examples and the associated individuals which define each can be described by various atomic concepts, their negations, and quantified roles.
- **Classify Local TBoxes.** Once the context graph is constructed, so-called local domains (§4.2, Definition 4.2.5) which were attributed throughout the context graph are used to define context-specific interpretations (§4.2.1, Definition 4.2.8). These are then used to deduce axiomatic knowledge about the relationship of various concept expressions in each context.
- **Prune Context Graph.** After the computation of local axioms for each context, portions of the context graph may be found to be redundant or irrelevant (§4.2.3) and are pruned in preparation for learning.
- **Start Learning Process.** The learning process runs Algorithm 7 until a set of solution concepts are found. As part of this process, a refinement operator such as $\rho_{\bar{\lambda}}$ is used to traverse the space of concepts constrained by the context graph.
- **Output OWL Classes.** If any solutions are found in the previous step, they are

converted from DL concepts to OWL class expressions and are associated with each of the individuals they describe, along with annotations describing their performance in terms of the selected measure function (e.g. accuracy, χ^2, etc.).

The OWL-Miner system is configurable to permit users to impose particular *declarative biases* which control various aspects of how the system will behave at runtime, including *language bias* to control the expressivity of the hypothesis language as follows:

- Limits on overall expression length or the maximum depth of any nested role expressions;
- The use or exclusion of any concept or role names;
- The use of disjunction, and if permitted, the maximum number of operands in any disjunction;
- The maximum number of occurrences of a role name appearing in any quantified role expression as operands to a conjunction;
- Use of any role quantifiers in addition to $\exists$, namely any of: $\forall$, ${}^{\geqslant n}$, ${}^{\leqslant n}$;
- Minimum and maximum cardinality of any qualified cardinality restrictions over particular role names globally;
- Use of the negation symbol against any atomic concept names.

The user is then able to describe the type of problem being solved as follows:

- Whether the problem is a classification or subgroup discovery problem, along with the convex measure function;
- The set of examples and their corresponding labels.

Once the type of problem is defined, *search bias* can be controlled as follows:

- Whether to use a best-first or stochastic beam search method, along with a limit on beam or expansion set size;
- Values for use in a heuristic utility function, such as α, β, γ on $\mathbf{u}_{OM}$;
- The maximum number of seconds to execute the search algorithm;

Lastly, the *validation bias* is controlled by defining the stopping criterion with:

- A minimum threshold on the convex measure function for recognising solutions and optimistic upper bounds;
- Whether the search algorithm should terminate as soon as k solutions have been found, or continue processing for the maximum specified time in an attempt to improve a top-k list of solutions.

Various other system-specific parameters can be controlled, including the maximum

Java Virtual Machine (JVM) memory limits and the maximum number of threads to use in refining and evaluating concepts in parallel in the instance chase and search algorithms. OWL-Miner also supports the automated execution of k-fold stratified cross-validation where $k > 1$ for use in assessing the performance of the concepts it generates as solutions.

The OWL-Miner system is open-source, and is available from GitHub at: https://github.com/owlminer/owl-miner.

6.2 Evaluation over Supervised Learning Problems

In this section, we describe the application of OWL-Miner to several well-known problems in classification and subgroup discovery which involve structured data. We compare the performance of OWL-Miner in terms of both quality of solutions found and system performance with DL-Learner, a state-of-the-art DL learning system, as well as reported results in ILP for the various problems.

Each problem described in this section was executed on a Dell PowerEdge® R820 with four Intel® Xeon® E5-4640 CPUs and 750Gb RAM. However, each problem that was run on this machine by OWL-Miner version 1.0.0 or DL-Learner version 1.2 was performed within a Java 8 JVM configured to use at most 16Gb heap space with at most one CPU thread, unless otherwise specified.

6.2.1 Michalski Trains

The Michalski Trains is a widely used dataset for testing learning systems which operate over structured data [64]. This dataset consists of only ten examples of trains and their features, such as aspects of their cargo, and are divided into two labels (eastbound and westbound), as depicted in Figure 6.2.

The OWL ontology describing trains consists of features such as the cars of each train, the shape of each car, the loads of each car, and the shape of each load, along with numerical counts of wheels per train and car, and load per car. Using the full expressivity of a DL hypothesis language, OWL-Miner was applied to classify both eastbound and westbound trains separately, which took a total of less than 1 second each to locate many different concepts of 100% accuracy. Some of the shortest

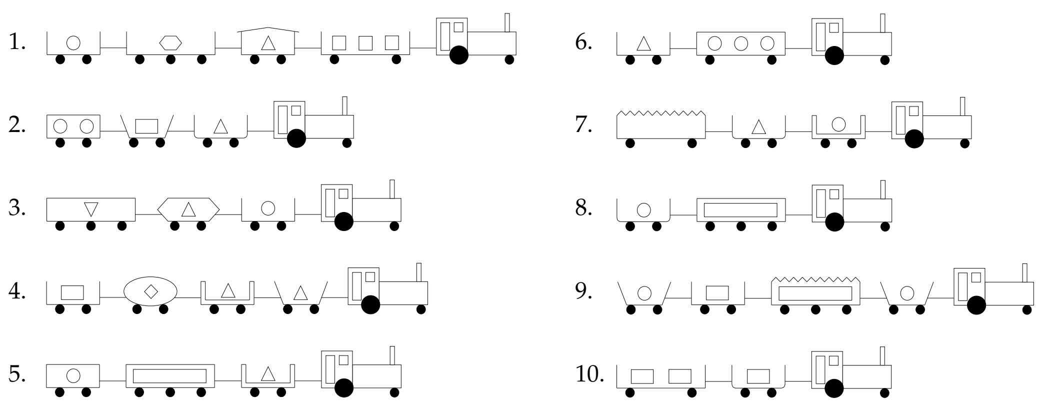


Figure 6.2: Michalski Trains. Trains numbered 1-5 are classified as belonging to the class 'eastbound', and those numbered 6-10 are labelled 'westbound'.

concepts describing eastbound and westbound trains were as follows:

Label	#	Expression
East	1	$Train \sqcap \exists hasCar.(Short \sqcap Closed)$
	2	$(Train \sqcap \forall hasCar.(Short)) \sqcup (Train \sqcap {}^{\geqslant 3}.hasCar.(Car \sqcap \exists.hasShape.(Load)))$
West	3	$(Train \sqcap {}^{\leqslant 2}hasCar.(Car)) \sqcup (Train \sqcap \exists hasCar.(Jagged))$
	4	$(Train \sqcap {}^{\leqslant 1}hasCar.(Short)) \sqcup (Train \sqcap \exists hasCar.(Jagged))$

These concepts can be read as: *eastbound trains are those which:*

1. *have at least one car which is short and closed;*
2. *only have short cars, or have at least three cars with the shape of a load,*

where $Load \equiv \{circle, rectangle, hexagon, triangle\}$ in the background ontology, and where *westbound trains are those which:*

3. *have at most two cars, or at least one car with a jagged roof;*
4. *have at most one short car, or at least one car with a jagged roof.*

The OWL-Miner system and the methods it implements have been designed to tackle learning problems which consist of a large amount of background knowledge and example data and which require a complex hypothesis language. While the Michalski Trains dataset is small, containing only a few concepts, roles and examples, it is nevertheless worthwhile demonstrating that OWL-Miner achieves results which are expected for this well-known problem.

6.2.2 Poker

The Poker dataset captures a structured representation of various five-card hands which are labelled with their corresponding poker hand type, such as *nothing*, *straight*, *pair*, *flush*, etc. [15]. The dataset presents a classification problem where a learning system must infer the definition of each of the poker hands to the exclusion of others.

This problem is interesting because it has proven to be very difficult for many learning systems, primarily because of the scale of the problem in the number of examples in the dataset, but also because it requires learned theories to be expressive in order to describe poker hands with high accuracy. The original dataset consists of a large number of examples of five-card poker hands, where each example is described only with the rank and suit of each card, for example:

A♠ *K*♠ *Q*♠ *J*♠ 10♠	(royal flush)	3♦ 4♦ 5♦ 6♦ 7♦	(straight flush)
7♦ 7♠ 7♣ 7♥ 3♦	(four of a kind)	*Q*♠ *Q*♣ *Q*♥ 9♠ 9♥	(full house)
J♣ 10♣ 8♣ 3♣ 2♣	(flush)	6♥ 5♦ 4♥ 3♥ 2♠	(straight)
5♦ 5♣ 5♠ *K*♦ 7♠	(three of a kind)	4♥ 4♦ *K*♠ *K*♥ 3♠	(two pair)
9♥ 9♠ 10♦ 4♦ 2♠	(one pair)	*K*♦ *Q*♠ 6♣ 7♠ 3♦	(nothing)

A translation of a portion of the poker dataset to an OWL ontology is distributed with the DL-Learner system. In this ontology, background knowledge has been added to aid in classification. Specifically, the roles *sameSuit*, *sameRank*, *nextRank* are used to assert whether cards within an individual hand have the same suit (e.g., 4♠ *sameSuit* 6♠), the same rank (e.g., *A*♦ *sameRank* *A*♣), or the next rank (e.g., 10♠ *nextRank* *J*♥).

The poker dataset is divided into a training set consisting of 25,010 examples across all hand types representing 0.00008% of all $311,875,200$ possible hands, and a test set consisting of one million examples. In testing OWL-Miner, we have sampled the training dataset alone and have created nine classification problems with 4,000 examples sampled across all classes consisting of a total of 22,307 individuals overall. Each of these nine problems treats examples of one known poker hand as the set of positive examples, with the remaining set of examples as negatives which encompasses examples of all other hands including those labelled with 'nothing'. In this way, we require that hypotheses are generated which correctly identify individual poker hands to the exclusion of all others, a classification approach in multi-class classification known as *one-versus-all*. We have added this data into a new ontology which incorporates concepts for describing each rank and suit, as well as roles for describing the cards of a hand, and rank and suit of a card, and the aforementioned three roles *sameSuit*, *sameRank* and *nextRank* to capture relationships between cards within each hand, as was incorporated into the poker OWL ontology distributed with the DL-Learner system. We ran OWL-Miner over this new dataset for each of the nine hand types, and for each, a concept with 100% accuracy was located in under 20 minutes of computation time as summarised in Table 6.1. In comparison, DL-Learner was also able to produce 100% accurate concepts for six of the nine hand types over the same data set, but failed to reach 100% for three classes within 60 minutes of computation time as shown in Table 6.2.

Hand	**Concept**	**Acc (%)**
One pair	$Hand \sqcap {\geqslant^2} hasCard.(Card \sqcap \exists sameRank.(Card)) \sqcap {\leqslant^2} hasCard.(Card \sqcap \exists sameRank.(Card))$	100.00
Two pair	$Hand \sqcap {\geqslant^4} hasCard.(Card \sqcap \exists sameRank.(Card) \sqcap {\leqslant^1} sameRank.(Card)) \sqcap {\leqslant^4} hasCard.(Card \sqcap \exists sameRank.(Card) \sqcap {\leqslant^1} sameRank.(Card))$	100.00
Three of a kind	$Hand \sqcap {\geqslant^3} hasCard.(Card \sqcap \exists .sameRank.(Card)) \sqcap {\leqslant^3} hasCard.(Card \sqcap \exists .sameRank.(Card))$	100.00
Straight	$Hand \sqcap \exists hasCard.(Card \sqcap {\leqslant^3} sameSuit.(Card) \sqcap \exists nextRank.(Card \sqcap \exists nextRank.(Card \sqcap \exists nextRank.(Card \sqcap \exists nextRank.(Card)))))$	100.00
Flush	$Hand \sqcap {\geqslant^4} hasCard.(Card \sqcap {\geqslant^4} sameSuit.(Card \sqcap {\leqslant^3} sameSuit.(Card) \sqcap \exists nextRank.(Card)))$	100.00
Full house	$Hand \sqcap {\geqslant^4} hasCard.(Card \sqcap \exists sameRank.(Card)) \sqcap {\leqslant^2} .hasCard.(Card \sqcap {\leqslant^1} sameRank.(Card))$	100.00
Four of a kind	$Hand \sqcap \exists hasCard.(Card \sqcap {\geqslant^3} sameRank.(Card))$	100.00
Straight flush	$Hand \sqcap \exists hasCard.(Card \sqcap {\geqslant^4} sameSuit.(Card \sqcap \exists nextRank.(Card \sqcap \exists hasRank.(\neg Ace))))$	100.00
Royal flush	$Hand \sqcap \exists hasCard.(Card \sqcap \exists hasRank.(Ace) \sqcap {\geqslant^4} sameSuit.(Card \sqcap \exists nextRank.(Card)))$	100.00

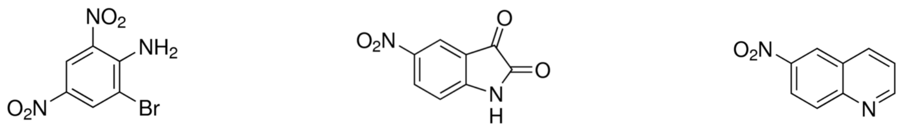
Table 6.1: Best concepts generated by OWL-Miner for each of the poker hand types for a maximum computation time of 20 minutes.

Hand	**Concept**	**Acc (%)**
One pair	$\exists hasCard.(\exists sameRank.(\leqslant^2 nextRank.(Thing)) \sqcap \leqslant^1 sameRank.(Thing))$	87.52
Two pair	$\geqslant^2 hasCard.(\exists hasSuit.(\neg Spades) \sqcap \exists sameRank.(\leqslant^1 sameRank.(Thing)) \sqcap \leqslant^2 nextRank.(Thing))$	85.34
Three of a kind	$\geqslant^2 hasCard.(\geqslant^2 sameRank.(\leqslant^1 sameRank.(\exists sameRank.(\exists sameSuit.(\exists sameRank.(Thing))))))$	100.00
Straight	$\exists hasCard.(\exists nextRank.(\exists nextRank.(\exists nextRank.(\exists nextRank.(\exists hasRank.(\neg Four \sqcap \neg Three \sqcap \neg Two)) \sqcap (\leqslant^3 sameSuit.(Thing))))))$	100.00
Flush	$\exists hasCard.(\exists nextRank.(\exists nextRank.(\exists nextRank.(\exists sameSuit.(\exists hasRank.(Four \sqcup King)) \sqcap (\geqslant^4 sameSuit.(\exists hasRank.(\neg Ten \sqcap \neg Three)))))))$	99.51
Full house	$\exists hasCard.(\geqslant^2 sameRank.(\exists sameRank.(\exists sameSuit.(\exists sameRank.(Thing)))))$	100.00
Four of a kind	$\exists hasCard.(\geqslant^3 sameRank.(Thing))$	100.00
Straight flush	$\exists hasCard.(\exists nextRank.(\exists nextRank.(\exists nextRank.(\geqslant^4 sameSuit.(\exists nextRank.(\exists hasRank.(\neg Ace)))))))$	100.00
Royal flush	$\exists hasCard.(\exists nextRank.(\exists nextRank.(\exists nextRank.(\exists nextRank.(\geqslant^4 sameSuit.(\exists sameSuit.(\exists hasRank.(Ace))))))))$	100.00

Table 6.2: Best concepts generated by DL-Learner with the OCEL search strategy for each of the poker hand types for a maximum computation time of 60 minutes.

6.2.3 Mutagenesis

Mutagenesis is a well-known benchmark problem in machine learning [24]. A variety of machine learning techniques have been applied to the mutagenesis dataset to construct classification models, from ILP to kernel-based methods [61], however the application of DL learning systems to the problem has not been described before. The mutagenesis dataset contains examples of various chemical compounds and their characteristics, such as the atomic structure including functional groups, and various real-valued measures such as a water/octanol partition coefficient, *log P*. The so-called 'regression friendly' dataset contains 188 example compounds, 125 of which are labelled positive for mutagenicity, and the remaining 63 are labelled negative. Figure 6.3 shows a sample of three compounds which appear in the mutagenesis dataset.



2-bromo-4,6-dinitroaniline 5-nitroisatin 6-nitroquinoline

Figure 6.3: Various small molecules from the mutagenesis dataset.

Originally, the mutagenesis dataset was represented in first-order predicate logic for use in ILP systems, but has also been converted to OWL for use in DL learning as is distributed with version 1.2 of the DL-Learner system. The resulting OWL ontology contains 88 classes, 5 object properties, 6 datatype properties and 14,145 individuals. The ontology contains classes which describe types of atoms, bond types and functional group structures.

We applied both the OWL-Miner and DL-Learner systems to this dataset [81], running each experiment in isolation on the same machine with 16Gb RAM and one CPU thread for a maximum runtime of 15 minutes. The same hypothesis concept language was used in both systems. Algorithm 7 was set to locate one best solution ($k = 1$) with beam (B_{max}) and expansion set (E_{max}) sizes of 10,000 each. During experimentation, we began testing with a minimum threshold on accuracy of 99%, but only found solutions at around 90% accuracy, which we set as the minimum accuracy for the experiments described below. In order to compare the performance of OWL-Miner with DL-Learner, we observed the accuracy of newly discovered best-performing candidates and the number of concepts which had been tested up to that point of discovery, as shown in Figure 6.4.

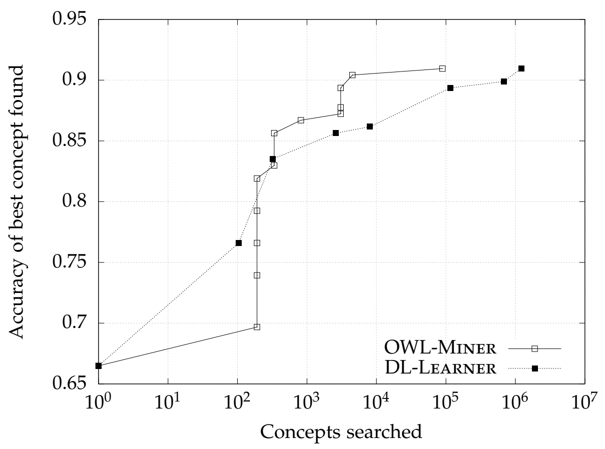


Figure 6.4: The performance of OWL-MINER and DL-LEARNER over the mutagenesis dataset, plotting the number of concepts searched by each system versus the accuracy of the best performing candidate.

From Figure 6.4, we see that OWL-MINER locates a concept with around 89% accuracy after searching through around 3,000 concepts, and eventually locates a concept with around 91% accuracy after searching through around 90,000 concepts. DL-LEARNER locates a concept of 86% accuracy after searching through around 8,000 concepts, and eventually locates a concept with around 91% accuracy after searching through over 1.2 million concepts, more than 13 times the number of concepts OWL-MINER took to reach a similar result. OWL-MINER located its best concept at 90.96% accuracy after less than 30 seconds, as follows:

$$\begin{aligned}&\mathit{Compound} \sqcap {\geqslant^4}\mathit{hasStructure}.(\neg \mathit{Methyl} \sqcap \neg \mathit{HeteroAromatic5Ring} \sqcap \\ &\neg \mathit{HeteroAromatic6Ring} \sqcap \neg \mathit{Benzene}) \sqcap {\geqslant^4}\mathit{hasAtom}.(\mathit{Hydrogen}_3) \sqcap \\ &\exists \mathit{lumo}.[\geq -3.768 \wedge \leq -1.102]\end{aligned}$$

This concept can be read as: *mutagenic compounds are those with at least four structures which are not methyl, benzene or hetero-aromatic 5 or 6 rings, and which have at least four hydrogen-3 atoms, and which have a lumo value of between -3.768 and -1.102.* Similarly, the best concept produced by DL-LEARNER also with an accuracy of 90.96% was:

$$\begin{aligned}&\mathit{Compound} \sqcap \exists \mathit{hasAtom}.(\exists \mathit{charge}.[\leq -0.368]) \sqcap \\ &{\geqslant^4}\mathit{hasStructure}.(\neg \mathit{Benzene} \sqcap \neg \mathit{Methyl}) \sqcap \exists \mathit{logp}.[\geq 1.91]\end{aligned}$$

This concept can be read as: *mutagenic compounds are those with at least one atom with a charge of less than or equal to -0.368, and which has at least four structures which are not* *methyl or benzene, and which has a logp value of at least 1.91.* Both of these concepts are expressive and easily comprehensible, highlighting the applicability of DL learning as a suitable method for this problem.

To test if the best concepts generated by OWL-Miner were over-fitting the data, we computed the 10-fold cross validation accuracy and F_1 scores for several minimum accuracy thresholds τ_{min} as shown in Table 6.3. We compared this to the best 5-fold[2] results produced by DL-Learner, and clearly see OWL-Miner produces a significantly stronger result.

System	τ_{min}	*Acc.* $\pm\,\sigma$ (%)	$F_1 \pm \sigma$ (%)	Runtime $\pm\sigma$ (s)	Length $\pm\sigma$
OWL-Miner	0.88	86.50 ± 0.09	89.34 ± 0.08	0.79 ± 0.81	11.4 ± 0.97
	0.89	88.25 ± 0.06	91.20 ± 0.04	13.19 ± 34.92	12.5 ± 2.01
	0.90	$\mathbf{90.50 \pm 0.09}$	$\mathbf{92.25 \pm 0.07}$	77.09 ± 104.21	15.7 ± 1.25
DL-Learner	0.90	84.67 ± 0.11	88.92 ± 0.06	916.33 ± 26.65	20.8 ± 5.68

Table 6.3: Cross validation accuracy and F_1 scores for various minimum thresholds over accuracy for the mutagenesis dataset with the OWL-Miner and DL-Learner systems.

A sample of previously reported best accuracies of a variety of methods which have been applied to the mutagenesis problem can be found in Table 6.4 from Lodhi and Muggleton [61], to which we have added our results for OWL-Miner, DL-Learner and also Alchemy, a higher-order logic learning system [72]. From Table 6.4 we see that the 10-fold cross validation accuracy for the mutagenesis problem ranges from around 85% to 95% accuracy, so the best accuracy produced in our experiments by OWL-Miner are comparable.

The strongest reported result for mutagenesis was produced by Aleph, an ILP-based system [61]. To obtain this result, Aleph generated twenty-five theories for an ensemble classifier, which differs greatly from the single-concept outputs of OWL-Miner and DL-Learner. We argue that such multi-clause theories are not as readily comprehensible as the individual DL concepts we have produced, and nevertheless observe that both OWL-Miner and DL-Learner could be modified to employ a similar ensemble strategy. We re-ran this experiment in our environment and found that Aleph took over 7 minutes of processing time to achieve its best result. We also note that Aleph was reported to have produced results of 86.3% to 87.7% accuracy in around 10 to 25 seconds [110], which is outperformed by OWL-Miner in terms of

[2]DL-Learner failed to complete a full 10-fold test, experiencing memory errors. We reduced the number of folds to test until the software successfully produced a result at 5-folds.

Type	System	Citation	Evaluation Method	Accuracy (%)
ILP	P-Progol	[95]	10-fold	88.0 ± 2.0
	FOIL	[77]	10-fold	86.7
	STILL	[89]	Single train/test split: 90/10	93.6 ± 4.0
PPILP	MFLOG	[50]	10-fold	95.7
	RSD		10-fold	92.6
	SINUS	[51]	10-fold	84.5
	RELAGGS		10-fold	88.0
EMILP	Aleph+RS	[61]	10-fold	**95.8 ± 3.3**
	Boosted FOIL	[77]	10-fold	88.3
Kernels	MIK	[34]	N/A	93.0
	RK	[22]	10-fold	85.4
	GK^3	[62]	leave-one-out	96.1
Naive Bayes + ILP	nFOIL	[53]	10-fold	78.3 ± 12.0
	Aleph+NB		10-fold	72.8 ± 11.7
Others	Neural Networks		10-fold	89.0 ± 2.0
	Linear Regression	[95]	10-fold	89.0 ± 2.0
	CART		10-fold	88.0 ± 2.0
HOL	ALCHEMY	[72]	10-fold	89.39
DLL	OWL-MINER		10-fold	90.50 ± 0.09
	DL-LEARNER		5-fold	84.67 ± 0.11

Table 6.4: Various accuracy results for the mutagenesis problem taken from [61], with the addition of Higher-Order Learning (HOL) from [72] and Description Logic Learning (DLL) reported for the first time here. The various other type acronyms are: Inductive Logic Programming (ILP), Propositionalisation-based ILP (PPILP), and Ensemble Methods in ILP (EMILP). The best result as highlighted in bold was achieved by an EMILP system based on Aleph [61].

scalability which produced a concept of 89% accuracy in around one second.

We attribute the strong performance of OWL-MINER over DL-LEARNER to the efficiency achieved by restricting the set of concepts available to the refinement operator which are suitable for testing as search candidates, together with the method of early pruning based on upper bound estimation, and the simultaneous learning of lower and upper bounds on datatype property restrictions with numerical ranges. These fundamentally different approaches are the main advantages the OWL-MINER system has over the DL-LEARNER system which enable it to locate high performing concepts efficiently.

6.2.4 Carcinogenesis

The carcinogenesis dataset is another long standing and well-known benchmark problem in machine learning [96]. Similar to the mutagenesis problem, the carcinogenesis dataset[3] contains structured examples of chemical compounds together with the results of various bioassays and are labelled as being carcinogenic or not. The full dataset contains 337 example compounds, 182 of which are labelled positive for carcinogenicity, and the remaining 155 are labelled negative as being non-carcinogenic. The OWL ontology capturing this dataset from the DL-Learner project contains 142 classes, 18 object properties, 1 datatype property and 22,374 instances, along with class axioms describing the subsumption hierarchy of atom, bond and functional group structural classes, such as various types of halides or ring structures.

The DL-Learner system has been reported to achieve the best results over the carcinogenesis dataset as a classification problem when setting a minimum accuracy of 72% [58]. In our first experiment, we compared the number of concepts searched by OWL-Miner when running a classification task based on accuracy with a minimum threshold of 72% against the performance of the current best concept. We also ran the same experiment with DL-Learner and the OCEL search strategy on the same machine, the results of which are shown in Figure 6.5. These results show that while OWL-Miner took slightly longer to locate high performing concepts initially, ultimately it searched fewer concepts (1,516,184) to reach a solution with the greatest overall accuracy of 71.51% after around 15 minutes of computation time. In comparison, DL-Learner found a concept of accuracy 70.33% after searching 2,028,625 concepts in around the same time. As previously reported [58], the DL-Learner system is indeed capable of locating high accuracy concepts for this problem. The concepts produced by OWL-Miner were similar to those found by DL-Learner, however OWL-Miner was generally faster.

The best concept produced by OWL-Miner for this experiment with an accuracy of 71.513% was:

$$\begin{aligned} &\mathit{Compound} \sqcap (\geqslant^4 \mathit{hasStructure}.(\mathit{Halide}) \sqcup (\forall \mathit{hasAtom}.(\neg \mathit{Sulfur}_{70} \sqcap \neg \mathit{Sulfur}_{75} \sqcap \\ &\quad \neg \mathit{Sulfur}_{76} \sqcap \neg \mathit{Titanium}_{134} \sqcap \neg \mathit{Nitrogen}_{31} \sqcap \neg \mathit{Nitrogen}_{35} \sqcap \neg \mathit{Nitrogen}_{36} \sqcap \\ &\quad\quad \neg \mathit{Oxygen}_{42}) \sqcap \exists \mathit{amesTestPositive}.(\{\mathit{true}\}))) \end{aligned}$$

The best concept produced by DL-Learner for this experiment with an accuracy

[3]http://www.cs.ox.ac.uk/activities/machlearn/cancer.html

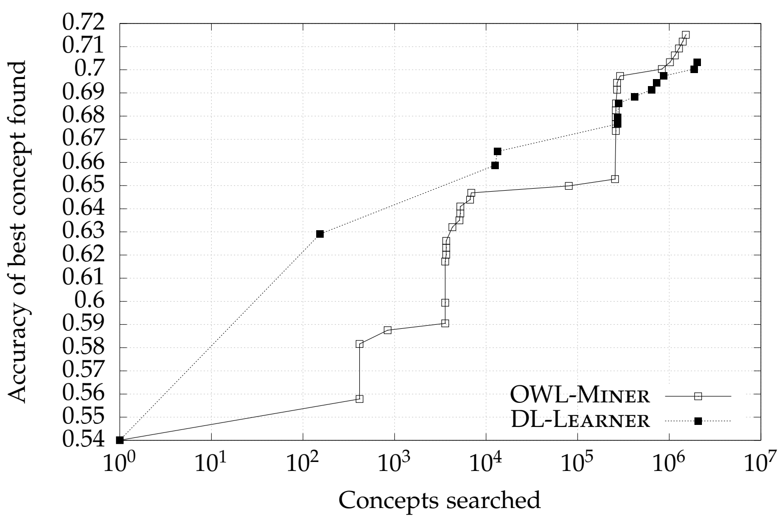


Figure 6.5: The performance of OWL-Miner and DL-Learner over the carcinogenesis dataset, plotting the number of concepts searched by each system versus the accuracy of the best performing candidate over a maximum runtime of 15 minutes.

of 70.33% was:

$$
\begin{aligned}
&(\geqslant^3 \mathit{hasStructure}.(\mathit{Halide} \sqcap \neg \mathit{Halide}_{10}) \sqcup \exists \mathit{amesTestPositive}.(\{\mathit{true}\})) \sqcap \\
&\exists \mathit{hasStructure}.(\neg \mathit{Phenol} \sqcap \neg \mathit{Ring}) \sqcap \\
&\quad \geqslant^2 \mathit{hasAtom}.(\neg \mathit{Iodine} \sqcap \exists \mathit{charge}.(\mathtt{double}[\leq -0.027]))
\end{aligned}
$$

The 10-fold cross validation accuracy of OWL-Miner for the carcinogenesis problem is summarised in Table 6.5. In producing these results, OWL-Miner was configured to determine the first top-10 concepts which met the minimum accuracy threshold as specified in each row of the table. For each fold, the concept with the highest accuracy over the training set was selected, which did not necessarily have the greatest test set accuracy. This is particularly apparent when the maximum error was set to 0.36 and 0.35. When the maximum error was set to 0.30, we find that OWL-Miner produced the highest reported 10-fold accuracy for this problem. These figures agree with those reported for DL-Learner, where it is observed that longer concepts found with a maximum error of 0.30 or less do not necessarily reduce the cross-validation test error, which may be a sign of over-fitting. Figure 6.6 plots the training and related test accuracies for this experiment.

Previously reported results for the carcinogenesis problem can be found summarised in Table 6.6 from [58], with the addition of the result for OWL-Miner.

Error	*Acc.* $\pm \sigma$ (%)	$F_1 \pm \sigma$ (%)	Runtime (s)	Length
0.40	61.27 ± 6.45	56.69 ± 10.26	0.21 ± 0.14	4 ± 0.00
0.39	62.24 ± 8.13	59.31 ± 12.17	0.17 ± 0.09	4 ± 0.00
0.38	62.31 ± 7.44	59.22 ± 12.03	0.09 ± 1.79	4.8 ± 1.68
0.37	60.99 ± 7.47	57.89 ± 7.64	11.54 ± 21.27	6.5 ± 2.42
0.36	59.81 ± 7.17	57.41 ± 11.13	35.61 ± 71.61	9 ± 2.40
0.35	57.84 ± 9.04	59.12 ± 10.03	158.41 ± 190.48	10.4 ± 1.26
0.34	63.96 ± 9.43	64.08 ± 13.10	237.49 ± 147.04	10 ± 2.62
0.33	65.44 ± 8.90	67.78 ± 8.35	902.62 ± 1374.95	12.1 ± 5.04
0.32	63.57 ± 11.13	65.17 ± 11.72	667.56 ± 872.74	12.9 ± 2.73
0.31	67.12 ± 8.72	68.50 ± 8.41	456.01 ± 324.93	13 ± 1.89
0.30	**69.04** ± **7.51**	**70.28** ± **7.52**	1659.21 ± 1110.36	16.9 ± 2.51
0.29	66.36 ± 4.95	67.42 ± 5.52	1706.72 ± 922.55	16.8 ± 2.35
0.28	67.17 ± 9.32	67.96 ± 9.82	2317.14 ± 1040.87	18.4 ± 3.37

Table 6.5: 10-fold stratified cross validation accuracy for the carcinogenesis problem for OWL-Miner.

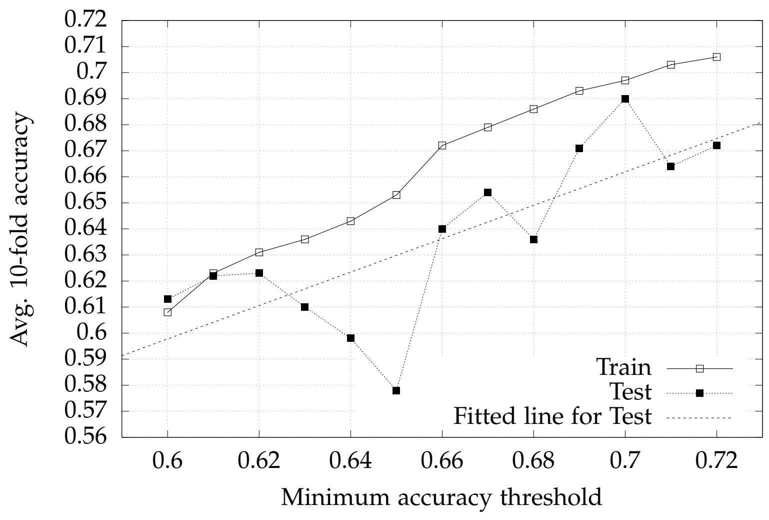


Figure 6.6: The 10-fold stratified cross validation performance of OWL-Miner for various minimum accuracy (maximum error) values over the carcinogenesis dataset, where we see from the fitted line that test accuracy mostly increases with training accuracy.

6.2.5 Mushrooms

The mushroom dataset [87] consists of 8,124 hypothetical examples of the characteristics of mushrooms from the *agaricus* and *lepiota* families, with roughly half labelled as being edible (4,208) and the other half as being poisonous (3,916).

Type	System	Citation	Accuracy (%)
ILP	Aleph with Ensembles	[27]	59.0 to 64.5
	Boosted Weak ILP	[44]	61.1
	Weak ILP	[44]	58.7
	Aleph Deterministic Top-Down 0.7	[110]	57.9 ± 9.8
	Aleph Randomized Rapid Restarts 0.9	[110]	57.6 ± 6.4
	Aleph Deterministic Top-Down 0.9	[110]	56.2 ± 9.0
	Aleph Randomized Rapid Restarts 0.7	[110]	54.8 ± 9.0
DLL	DL-Learner	[58]	*67.4 ± 7.9*
	OWL-Miner		**69.04 ± 7.51**

Table 6.6: Various accuracy results for the carcinogenesis problem from [58], with the inclusion of DL-Learning (DLL) results for OWL-Miner and DL-Learner systems. The best result as highlighted in bold was achieved by OWL-Miner and second-best in italics with DL-Learner.

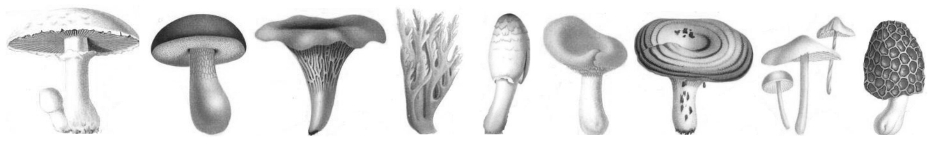


Figure 6.7: The mushroom dataset [87] contains feature descriptions of thousands of mushrooms, labelled as being either edible or poisonous (image from [102]).

Originally, this dataset was presented in an attribute-value format, but for the purposes of evaluating the performance of OWL-Miner we have converted the data into RDF and OWL. As part of this process, mushrooms are described by their components and sub-components, such as a mushroom having a cap and stalk, which each have different shapes, surface types, colors, and gills. The resulting ontology contains 84 classes, 19 object properties, 2 datatype properties and 40,679 individuals describing each of the examples.

We executed both OWL-Miner and DL-Learner over this problem to learn single concepts which correctly classify edible mushrooms over poisonous ones, as a simple known concept already exists for this problem, namely: *edible mushrooms are those with a spore print color which is not green, and which have an odor which is not almond or anise.* As this concept has 99.41% accuracy on the whole dataset, we set the maximum error rate for concepts learned by either system to be 1%.

Figure 6.8 plots the runtime execution of OWL-Miner and DL-Learner for this problem, where we observed that OWL-Miner found the highest accuracy concept at 99.41% in 42 seconds after testing less than 5,000 unique concepts. In comparison, DL-Learner using the OCEL search strategy finds concepts of at most 98.82%

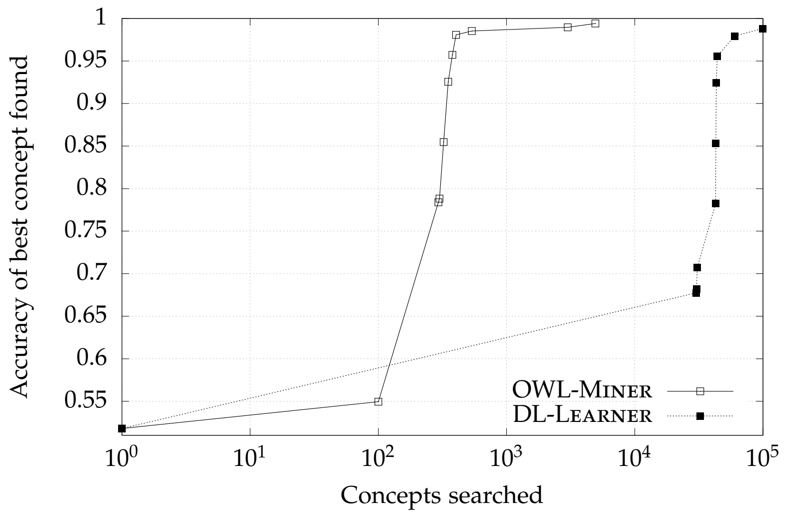


Figure 6.8: The performance of OWL-Miner and DL-Learner over the mushroom dataset, plotting the number of concepts searched by each system versus the accuracy of the best performing candidate.

accuracy after testing around 100,000 concepts taking more than 20 minutes of computation time on the same machine. Afterwards the system was allowed to continue running for a total of 30 minutes but no better concept was found after testing 174,000 concepts and the search was terminated. Similarly, executing DL-Learner with the CELOE search strategy resulted in the best concept having 98.08% accuracy also after 30 minutes of execution time, which is poorer so is not depicted in Figure 6.8. The best concept produced by OWL-Miner was:

$$\begin{aligned}&\mathit{Mushroom} \sqcap \exists \mathit{hasOdor}.(\neg \mathit{Foul} \sqcap \neg \mathit{Fishy} \sqcap \neg \mathit{Spicy} \sqcap \neg \mathit{Pungent} \sqcap \neg \mathit{Creosote}) \sqcap \\ &\quad \sqcap \exists \mathit{hasSporePrintColor}.(\neg \mathit{Green})\end{aligned}$$

This concept has an accuracy of 99.41% and enumerates the odors which indicate poisonous mushrooms, and is therefore similar to the most general rule with the same accuracy as described in the literature.

The best concepts produced by DL-Learner were:

$$\begin{aligned}&\mathit{Mushroom} \sqcap \exists \mathit{hasOdor}.(\neg \mathit{Foul} \sqcap \neg \mathit{Fishy} \sqcap \neg \mathit{Spicy} \sqcap \neg \mathit{Pungent} \sqcap \neg \mathit{Creosote}) \sqcap \\ &\quad \exists \mathit{hasRing}.(\exists \mathit{ringNumber}.(\mathtt{int}[\geq 2]) \sqcap \exists \mathit{hasSporePrintColor}.(\neg \mathit{Green})\end{aligned}$$

for the OCEL search strategy with an accuracy of 98.82%, and:

$$\begin{aligned}&\mathit{Mushroom} \sqcap \exists \mathit{hasOdor}.(\neg \mathit{Foul} \sqcap \neg \mathit{Fishy} \sqcap \neg \mathit{Spicy} \sqcap \neg \mathit{Pungent} \sqcap \neg \mathit{Creosote}) \sqcap \\ &\quad \exists \mathit{hasRing}.(\neg \mathit{Large})\end{aligned}$$

for the CELOE search strategy with an accuracy of 98.08%. It is possible that concepts without refinements of the property *hasRing* were rejected in the local search implemented by DL-Learner, as otherwise it is clear it should have located the same best concept as found by OWL-Miner. We also tested the 10-fold stratified cross-validation performance of OWL-Miner of this problem, which achieves a 99.22% ± 0.04% accuracy and 99.23% ± 0.001% F1 score.

If we assume the mushroom dataset is a representative sample of all species of mushrooms from the *agaricus* and *lepiota* families, we may be equally as interested in high accuracy concepts which describe poisonous mushrooms. This assumption may be contrast with the carcinogenesis and mutagenesis datasets which sample a tiny proportion of all possible molecules, where learning concepts to describe non-carcinogenic or non-mutagenic molecules would be generally less useful. By configuring OWL-Miner to seek concepts which classify poisonous mushrooms, it produced the following concept with 99.41% accuracy within 15 minutes:

$$\begin{aligned}&(\mathit{Mushroom} \sqcap \exists \mathit{hasRing}.(\mathit{Pendant}) \sqcap \exists \mathit{hasSporePrintColor}.(\mathit{Green})) \sqcup \\ &\quad (\mathit{Mushroom} \sqcap \exists \mathit{hasOdor}.(\neg \mathit{Nil} \sqcap \neg \mathit{Anise} \sqcap \neg \mathit{Almond}))\end{aligned}$$

We also configured and ran DL-Learner to perform the same learning task which produced the following concept with 98.52% accuracy using the OCEL search strategy after 2 minutes, with no better concept found after searching for a further 58 minutes:

$$\mathit{Mushroom} \sqcap \exists \mathit{hasOdor}.(\neg \mathit{Nil} \sqcap \neg \mathit{Anise} \sqcap \neg \mathit{Almond})$$

Lastly, to construct concepts which describe either poisonous or edible mushrooms, we ran OWL-Miner to solve this as a subgroup discovery problem with a minimum threshold of 90% weighted relative accuracy (Definition 3.3.4) for a maximum of 15 minutes. The results of this single experiment are summarised in Table 6.7. From this table, we see that a variety of features of mushrooms can be used to describe edible or poisonous subgroups with high accuracy. Because the dataset is hypothetical, in practice these rules should not be used to distinguish poisonous mushrooms from edible ones, however the experiment demonstrates the capability of the OWL-Miner system to generate different descriptive hypotheses of either type over a variety of features in a single experiment.

WRA %	**Acc** %	**Concept**
-96.936	1.477	$Mushroom \sqcap \exists hasOdor.(\neg Nil \sqcap \neg Anise \sqcap \neg Almond)$
-90.472	4.825	$Mushroom \sqcap \exists hasOdor.(\neg Nil \sqcap \neg Anise) \sqcap \exists hasCap.(Cap \sqcap \exists hasShape.(\neg Bell))$
-90.123	4.973	$Mushroom \sqcap \exists hasPopulation.(\neg Numerous \sqcap \neg Clustered) \sqcap \exists hasOdor.(\neg Nil \sqcap \neg Almond) \sqcap \exists hasSporePrintColor.(\neg Purple)$
96.936	98.523	$Mushroom \sqcap \exists hasOdor.(\neg Foul \sqcap \neg Fishy \sqcap \neg Musty \sqcap \neg Spicy \sqcap \neg Pungent \sqcap \neg Creosote)$
92.237	96.012	$Mushroom \sqcap \exists hasSporePrintColor.(\neg Green \sqcap \neg Chocolate) \sqcap \exists hasRing.(\neg Large) \sqcap \exists hasCap.(Cap \sqcap \exists hasGill.(Gill \sqcap \exists hasSize.(Broad)))$
91.317	95.569	$Mushroom \sqcap \exists hasRing.(\neg None \sqcap \neg Large) \sqcap \exists hasSporePrintColor.(\neg Chocolate) \sqcap \exists hasCap.(Cap \sqcap \exists hasGill.(Gill \sqcap \exists hasSize.(Broad)))$

Table 6.7: Several top concepts generated by OWL-Miner representing subgroups in the mushroom dataset using weighted relative accuracy (WRA) with a minimum threshold of 90%. Concepts with positive WRA values largely correspond to edible mushrooms, and negative WRA values largely correspond to poisonous mushrooms.

6.3 Performance Analysis

This section contains an analysis of the performance of novel algorithms implemented in the OWL-Miner system, including: the construction of the context graph and local axioms in preparation for learning (§6.3.1); the speed of refinement while learning (§6.3.2); detection of redundancy in learning with our refinement operator $\rho_{\bar{\lambda}}$ (§6.3.3); the speed of coverage checking (§6.3.5); and the effect of parallelisation in the construction of the context graph and in our learning algorithm (§6.3.6).

6.3.1 Preparing for Learning

In this section, we analyse the cost of construction of the context graph for each of the problems of Section 6.2. Prior to the execution of the learning Algorithm 7 in OWL-Miner, the knowledge base is pre-processed with classification by a DL reasoner, then a context graph (Definition 4.2.12) is constructed to describe the space of concept terms available to the refinement operator $\rho_{\bar{\lambda}}$. We assess the performance of Algorithm 4 which is used to construct a context graph in preparation for learning.

The performance of Algorithm 4 is a function of several parameters, including the size of the number of examples and the individuals and literals connected to them by role assertions in $\mathcal{A}$, the number of concepts describing these individuals, and the expressivity of the chosen hypothesis language. For each of the problems analysed in Section 6.2, Table 6.8 captures statistics about the time taken to compute the context graph in each case. As each context graph is shaped like a number of trees, this table also lists the total number of leaves corresponding to the contexts of the most deeply nested subexpressions to give an indication of the resulting size. The table also lists the number of examples, concepts, and roles and the chosen hypothesis expressivity for each problem. In each case, classification of the knowledge base took less than two seconds with the Pellet DL reasoner.

From Table 6.8a we see that the time taken to construct the context graph did not exceed 30 seconds for any problem when existential ($\exists$), universal ($\forall$) and minimum qualified cardinality restrictions ($^{\geqslant n}$) were included in the hypothesis language. The resulting context graph, which is structured like a number of trees rooted at every concept which describes any of the examples $e \in \mathcal{E}$, is shown in terms of the number of leaves which indicates the size of the graph by number of paths from the root of each tree. By simply including maximum qualified cardinality restrictions across all roles for each problem, we see from Table 6.8b that the time taken to construct the

Problem	**Time (s)**	**Leaves**	$\lvert\mathcal{E}\rvert$	$\lvert N_I\rvert$	$\lvert N_C\rvert$	$\lvert N_{OR}\rvert$	$\lvert N_{DR}\rvert$	$\geqslant^n r$
Poker	15.79	62,237	603	3,635	39	5	0	4
Muta.	16.61	15,221	230	14,145	87	5	6	5
Mush.	25.43	285	8,124	40,679	83	19	2	5
Carc.	29.49	20,086	337	22,374	144	18	1	5

(a) Size of the context graph and time to compute for a number of problems. The maximum value n for minimum qualified cardinality restrictions ($\geqslant^n r$) is shown.

Problem	**Time (s)**	**Leaves**	$\leqslant^n r$
Mush.	25.46	285	5
Muta.	41.00	45,188	5
Carc.	68.44	57,790	5
Poker	357.33	927,385	4

(b) As per Table 6.8a above, but also including maximum values n for maximum qualified cardinality restrictions ($\leqslant^n r$) in the hypothesis language.

Table 6.8: Context graph construction times with Algorithm 4 for a variety of problems along with the size of the resulting context graph viewed as a tree by the number of leaves, along with problem statistics and the use of existential ($\exists$), universal ($\forall$) and minimum qualified cardinality restrictions ($\geqslant^n$) in the hypothesis language. $\lvert N_{OR}\rvert$ and $\lvert N_{DR}\rvert$ denote the number of object and datatype roles, respectively.

context graph, along with the size of the resultant graph, increases by a factor of around 2 to 22 for each problem except the mushroom dataset.

The time taken to execute Algorithm 4 depends on the structure of the examples in the knowledge base, as well as the number of concepts, roles and role quantifiers available to describe these. Notably, the occurrence of r-successor sets (Definition 4.2.14) with cardinalities greater than 1 in combination with minimum and maximum cardinality restrictions generate many new edges between nodes in the context graph as the various possible cardinalities are enumerated. We observe this particularly in the Poker dataset (§6.2.2) where each individual representing a card may have multiple successors for the roles *sameRank* and *sameSuit* in each hand, thus causing Algorithm 4 to generate multiple edges between each class describing cards by suit and rank. This high branching factor is compounded by the depth of the context graph which permits nested sequences of roles by describing links between each card in a hand. Notably, while the Mushroom dataset (§6.2.5) had many more examples, individuals, classes and roles than the Poker dataset, context graph construction took less than 30 seconds to produce a several trees with 285 leaves in total, as each individual was often only linked to others by single role assertions. This limited the branching factor of the context graph as minimum and maximum cardinality restrictions were found to be irrelevant for use with the mushroom dataset and were subsequently excluded by Algorithm 4.

The time taken to construct the context graph compares favourably with the computation time of the learning Algorithm 7 in locating solutions for each of the problems described above. In each case, the time taken to compute the context graph in addition to the computation time for learning was less than the time taken by DL-Learner, and in each case, producing stronger results.

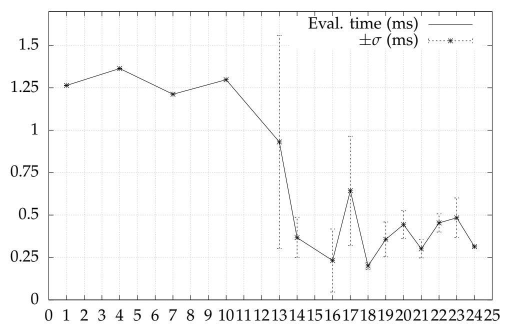
6.3.2 Refinement Evaluation Speed

Once the context graph is computed, it is used in the learning Algorithm 7 by the refinement operator $\rho_{\bar{\lambda}}$ to determine which concept expressions are available to refine to for any subexpression of a concept. As part of this process, the refinement operator traverses the context graph to determine the set of most applicable contexts (Definition 4.2.19) for every subexpression of a concept C under refinement. Figure 6.9 presents the various times taken for refinement of all subexpressions of a single concept for various concept lengths. In each problem displayed, the most expressive hypothesis language was used to assess the performance of the operator for the construction of as many concepts as possible.

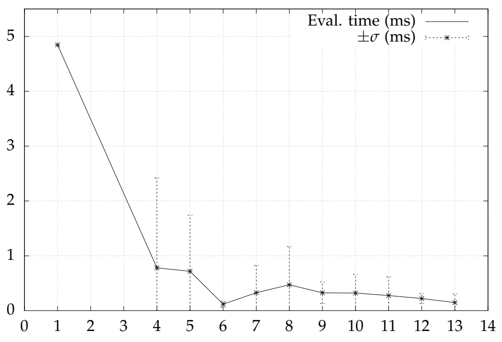
All values plotted in Figure 6.9 correspond to the time taken for a call of the refinement operator to produce *all* refinements of any single expression. As longer concepts have potentially more subexpressions to refine, we might reasonably expect the time taken to produce refinements of longer concepts to take longer, however the figures show otherwise. We attribute this result to the context graph which limits the number of options available to the refinement operator in deeply nested subexpressions which cover fewer individuals and literals as computed by the chase Algorithm 4.

6.3.3 Occurrences of Redundancy

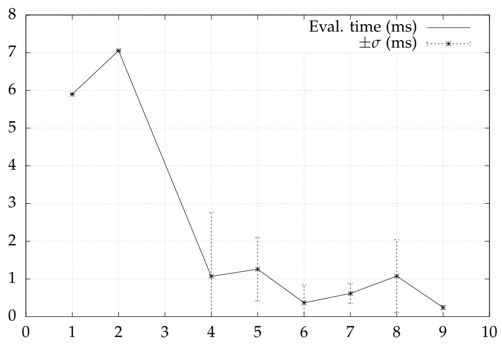
Proposition 4.5.3 describes how the refinement operator $\rho_{\bar{\lambda}}$ is *redundant* (Definition 3.4.8) in that the application of this operator may produce multiple refinement chains leading to equivalent concepts. This property is a source of inefficiency in the search performed by Algorithm 7 as the operator $\rho_{\bar{\lambda}}$ will direct the search into previously visited parts of the search space. In each of the problems discussed in Section 6.2, we analysed how many times a particular concept was generated in the



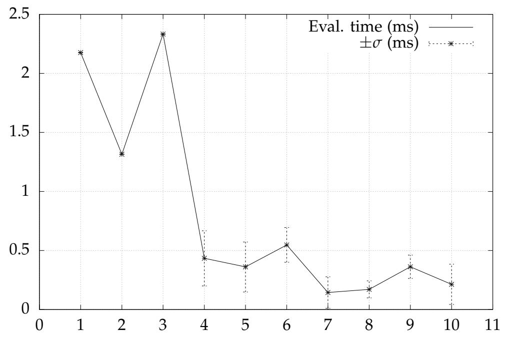
(a) Poker



(b) Mutagenesis



(c) Carcinogenesis



(d) Mushroom

Figure 6.9: Refinement speed versus concept length for all four benchmark problems discussed in Section 6.2. These figures show the average time in milliseconds (with standard deviation error) taken to produce all refinements of a single concept for concepts of various lengths.

search by maintaining a set of all concepts generated by $\rho_{\bar{\lambda}}$ and recognising when concepts were revisited.

From Figure 6.10 we see that redundancy in the search occurs in varying proportions based on the problem being solved. As the hypothesis language was largely the same for each problem analysed in Figure 6.10, the difference lies in the search space of concepts which is structured with respect to the input examples and the concepts and roles in the knowledge base which describe them. After analysis of the execution of these problems, a common source of redundancy appears to be refinement chains of the following form:

$$\begin{array}{lllll} \Diamond r.(A) & \rightsquigarrow_{\rho_{\bar{\lambda}}} & \Diamond r.(B) & \rightsquigarrow_{\rho_{\bar{\lambda}}} & \Diamond r.(B) \sqcap \Box s.(C) \\ \Diamond r.(A) & \rightsquigarrow_{\rho_{\bar{\lambda}}} & \Diamond r.(A) \sqcap \Box s.(C) & \rightsquigarrow_{\rho_{\bar{\lambda}}} & \Diamond r.(B) \sqcap \Box s.(C) \end{array}$$

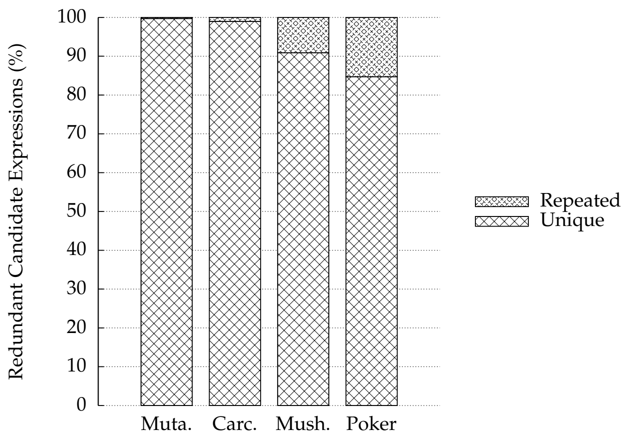


Figure 6.10: The proportion of repeated versus unique candidate expressions encountered in the entire search for solutions to various problems described in Section 6.2. Repeated expressions occur because of the redundancy of the refinement operator $\rho_{\bar{\lambda}}$ used in the search.

The operator $\rho_{\bar{\lambda}}$ does not restrict which type of refinement steps which are permitted in any one subexpression context, such as in this example with the refinement of the filler concept $A \rightsquigarrow_{\rho_{\bar{\lambda}}} B$ or conjunction with $\Box s.(C)$. As such, both of these steps are permitted as refinements of $\Diamond r.(A)$. For learning problems which may permit many different quantified role expressions in any one subexpression context, this particular source of redundancy appears to be more likely. For example, we observe that this is indeed the case with the Poker data set where each subexpression context referring to instances of *Card* can be refined in conjunction with several other quantified roles such as *nextRank*, *sameSuit* and *sameRank* which each have the range *Card*, thereby permitting nested expressions of the same kind. This situation results in a large number of possible redundant refinement chains as shown in Figure 6.10 which reports that around 15% of all concepts were revisited throughout the search for solutions.

In the OWL-Miner system, redundancy in the search is managed by normalising all concept expressions by ordering operands by their partial order, permitting the fast identification of syntactically equivalent concepts. In this way, the set of concepts in the search frontier or beam of Algorithm 7 can maintain unique sets of concepts at all times and eliminate duplicates. However, as the frontier or beam is fixed in size, the search may re-introduce previously seen concepts after they leave the frontier or beam set. For this reason, OWL-Miner is configurable to maintain a seen set for recording concepts which have been expanded by refinement to detect redundancy globally. This seen set permits the search to avoid refining a concept which has been refined before. The use of this seen set is optional however, as it potentially consumes a large amount of memory at runtime for long searches, and the redundancy of $\rho_{\bar{\lambda}}$ appears, at least by Figure 6.10, to be limited in practice.

6.3.4 Occurrences of Improper Refinements

Proposition 4.5.4 describes how the refinement operator $\rho_{\bar{\lambda}}$ is *improper* (Definition 3.4.5) in that, for any concept C, the application of this operator as $\rho_{\bar{\lambda}}(C)$ may produce refinements which are equivalent to C. As discussed in Section 5.3.2 of Chapter 5, improperness can negatively impact the performance of a learning algorithm which searches by refinement by introducing concepts into a limited search space which cannot be distinguished as better or worse relative to each other by heuristics based on coverage. In this situation, the search may prune away concepts which lead to solutions, or may at least waste computational resources in considering concepts which do not lead to solutions.

In each of the problems discussed in Section 6.2, we analysed how many times improper refinement steps occurred in the search versus all refinement steps. Furthermore, we compared this to the rate of improper refinement steps which occurred with the *subexpression suspension method* as described in Section 5.3.2 to limit so-called *ineffectual refinements* (Definition 5.3.1), the results of which are presented in Table 6.9 and Figure 6.11.

Problem	**Improper (%)**	**Improper SS (%)**	**Δ (%)**
Muta.	41.54	33.78	-18.68
Carc.	8.70	6.16	-29.20
Mush.	46.81	36.21	-22.64
Poker	41.55	19.58	-52.88
Trains	69.24	56.26	-18.75

Table 6.9: Proportions of occurrences of improper steps amongst all refinement steps for a variety of problems discussed in Section 6.2, along with the proportion of occurrences of improper refinement steps with *subexpression suspension* enabled (SS) (§5.3.2), and the difference between the two.

From Table 6.9 and Figure 6.11 we see that improper refinement steps in the search occur in varying proportions based on the problem being solved. Note that the hypothesis language was the same for each problem analysed in Figure 6.11. Improperness appears to account for a significant proportion of all refinement steps,

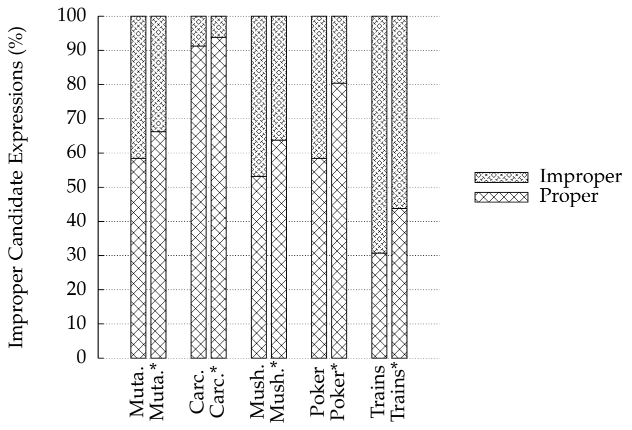


Figure 6.11: The proportion of (proper) versus (improper) refinement steps encountered in the entire search for solutions to various problems described in Section 6.2 shown both without and with (*) the subexpression suspension method. Improper refinements occur because of the improperness of the refinement operator $\rho_{\bar{\lambda}}$ used in the search.

and we observe that this is particularly the case when refining concepts which involve disjunctions as was enabled for each of the problems tested here where ineffectual refinements may occur. The results of various experiments reported in Section 6.2 were all conducted with subexpression suspension enabled. With subexpression suspension disabled, we observed that the amount of improper refinement steps increased by 18% to 52%, however generally still reached the same solutions in comparable time for these experiments in particular.

6.3.5 Cover Evaluation Speed

During the execution of the learning Algorithm 7, the two main operations are the refinement of candidate concept expressions with $\rho_{\bar{\lambda}}$, and their evaluation relative to certain performance measures which rely on coverage computation by Algorithm 11. In the previous section, we showed that refinement was generally very fast for a variety of problems. In this section, we analyse the performance of coverage checking by Algorithm 11 also relative to the same set of problems. From Figure 6.12, we observe that the computation of coverage for candidate hypothesis expressions by Algorithm 11 completely dominates the overall processing time of learning Algorithm 7 which includes concept refinement and all other operations. For each of these problems, we profiled the average time taken to compute coverage for individual concept expressions of varying lengths, and also by maximum role depth in the

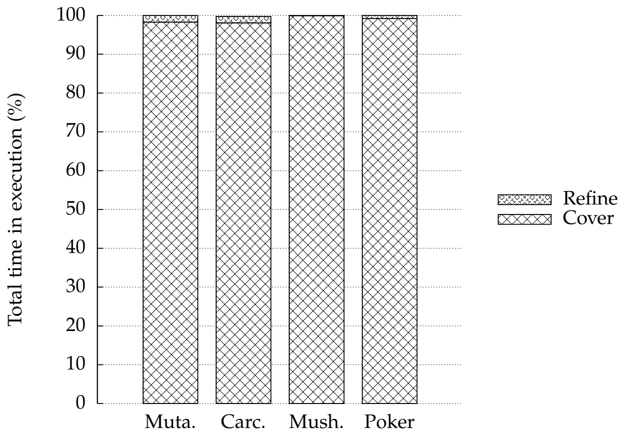
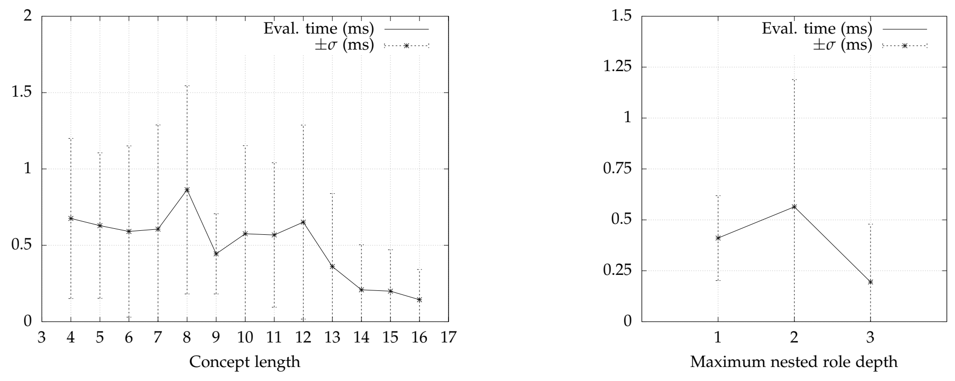


Figure 6.12: This plot shows the proportion of time spent in refinement (refine), and coverage evaluation (cover) for each of the various problems described in Section 6.2. We clearly see that coverage evaluation dominates processing time in each case, with refinement accounting for generally less than 2% of the total computation time for each problem.

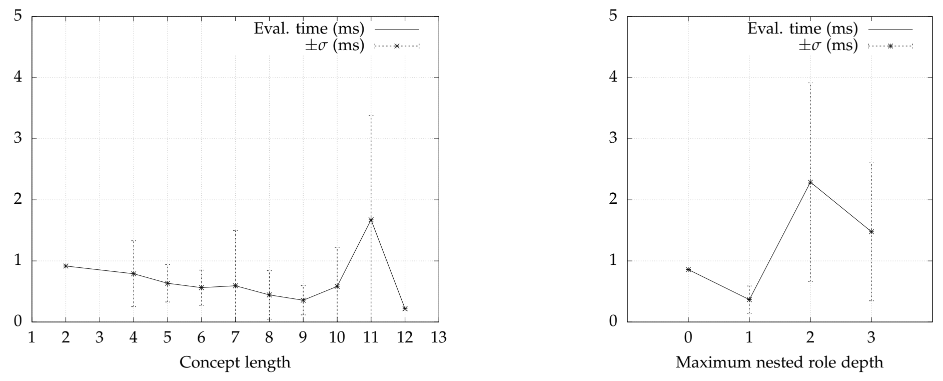
expression as this was shown to be the dominating factor in the complexity of this operation in Section 5.2.1.

From Figures 6.13 and 6.14 we observe that the overall processing time to fully evaluate the cover of each candidate hypothesis rarely exceeds several milliseconds. On the Carcinogenesis and Mutagenesis problems, complete evaluation generally took no longer than 1 millisecond. For Carcinogenesis, coverage computation involved testing up to 337 examples spanning a maximum of 22,374 instances and literals, and for Mutagenesis, coverage computation involved testing up to 230 examples spanning a maximum of 14,145 instances and literals. The Mushroom and Poker datasets contained considerably more examples, where Mushrooms contained 8,124 examples spanning a maximum of 40,679 instances and literals, and where Poker contained 4,000 examples spanning a maximum of 22,307 instances.

In general, we observe that as concepts grow in size, the time taken to evaluate their cover over all examples does not tend to increase significantly. We expect this is because longer concepts tend to pose more constraints on examples, permitting Algorithm 10 for instance checking to fail fast, which in turn permits Algorithm 11 to move on and check other examples. This effect may also apply to concepts with deeply nested roles, as we observed that the time taken to compute coverage for concepts with nested roles does not generally significantly increase with nested role depth.



(a) Mutagenesis



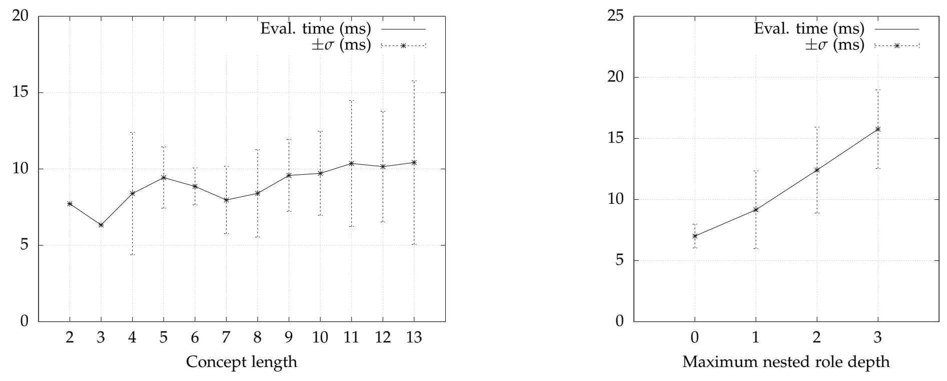
(b) Carcinogenesis

Figure 6.13: Average cover evaluation speeds for the Mutagenesis (6.13a) and Carcinogenesis (6.13b) datasets.

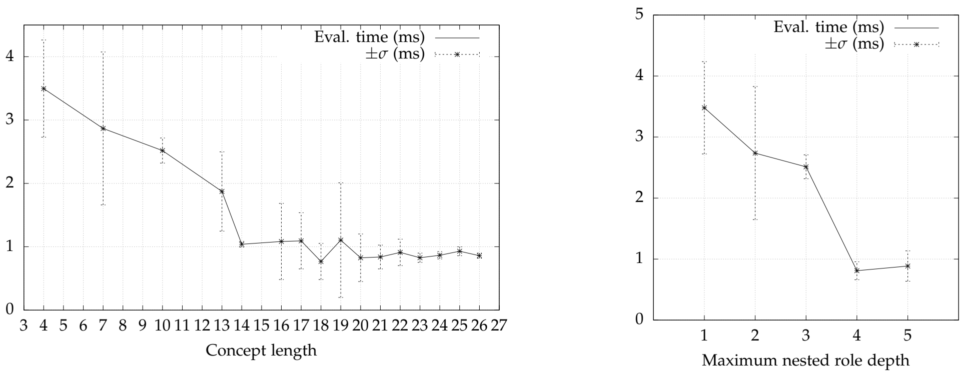
6.3.6 Parallelisation

Parallelism is exploited in two ways by the OWL-Miner system implementation, in both the computation of concept coverage, and in the instance chase procedure prior to learning. As we observed in Figure 6.12, it is clear that the majority of computation time spent in learning is dominated by concept coverage evaluation. Because of this, OWL-Miner parallelises the coverage evaluation of all refinements of a concept amongst a pool of worker threads. Once complete, each thread independently determines if the evaluated refinement should enter the frontier or beam, and if so, synchronously updates this set with the new candidate. Therefore, as the number of threads grows, adding new refinement candidates to the frontier or beam is the bottleneck as multiple threads will block until they gain access.

Figure 6.15 plots the performance of OWL-Miner for the Carcinogenesis prob-



(a) Mushroom



(b) Poker

Figure 6.14: Average cover evaluation speeds for the Mushroom (6.14a) and Poker (6.14b) datasets.

lem for various numbers of threads for coverage evaluation. As we can see, using multiple threads reduces the overall processing time to reach solutions of a certain accuracy, however is limiting in that as the number of threads grows, the longer they wait to lock the frontier or beam to add new candidates after evaluation. Using multiple threads to evaluate the cover of concepts and to add them to the frontier or beam in this way also introduces non-determinism in the search. This occurs because threads which update the frontier with new candidates may do so in an uncontrolled order, and as the frontier or beam is fixed in size, various candidates may be pruned over others simply based on the time at which they were introduced. In practice, we found that the non-determinism introduced by processing coverages in parallel was evident in the time taken to reach solutions but not in the quality of solutions, as the best candidates are the least likely to be pruned when ordered by the heuristic function.

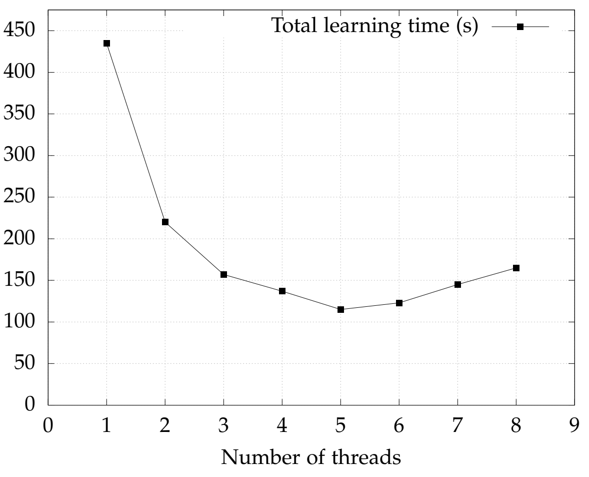


Figure 6.15: The effect of parallelisation on learning Algorithm 7. As more threads are available to perform coverage computation on refined expressions, the overall processing time to reach solutions of 69% accuracy for the Carcinogenesis problem is generally reduced, however increases slightly as more threads contend to synchronously update the single frontier or beam set with results.

The instance chase of Algorithm 4 is also inherently parallelisable as r-successor sets can be computed for each individual in the knowledge base independently, and the resulting context graph components can be updated synchronously.

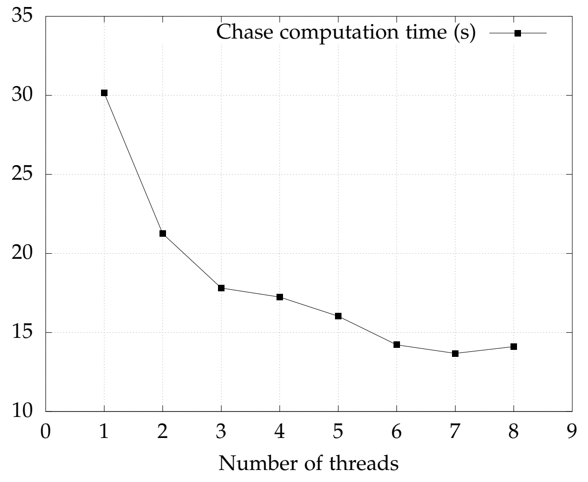


Figure 6.16: The effect of parallelisation on the instance chase Algorithm 10. As more threads are available to perform the instance chase, the overall processing time to compute the context graph for the Carcinogenesis problem is reduced.

Figure 6.16 plots the performance of OWL-Miner for the Carcinogenesis problem for various numbers of threads when performing the instance chase Algorithm 10 for constructing the context graph. As we can see, using multiple threads reduces the overall processing time, but as we saw when parallelising the learning problem, the performance increase is limited by the requirement to update the context graph synchronously as threads wait for a lock on parts of the context graph to update it. This effect appears to be less significant in this case however, as the context graph is a large, multi-object data structure and threads are more likely to update different graph components, reducing the possibility that each thread will need to block to wait for resources. With fine-grained locking, the instance chase algorithm is therefore more amenable to parallelisation. In Figure 6.16 we note that the overall computation time increased slightly when the number of threads was set to equal the maximum number available on the host machine, which in this case was set to eight. This can be explained by the fact that the main thread executing the overall algorithm was set to wait while other threads processed the context graph, and was not able to operate at full capacity as opposed to when the number of threads for processing the chase was less than eight.

6.4 Conclusion

In this section, we described our system OWL-Miner which implements the various algorithms discussed in this thesis, from the instance chase Algorithm 10 to the learning Algorithm 7 and supplementary functions such as the refinement operator $\rho_{\bar{\lambda}}$ and coverage checking Algorithm 11. We saw that OWL-Miner performs favourably against other state of the art implementations including DL-Learner which is the closest system for comparison. In particular, OWL-Miner produces strong results in terms of both the quality of solutions found as evidenced across four particularly challenging benchmark datasets: poker, mutagenesis, carcinogenesis and mushrooms. In practice, the OWL-Miner system has been developed with a particular goal in mind, which is to support the analysis of other similarly large and complex knowledge bases in the life sciences. In the next chapter, we will describe how we are working on integrating the OWL-Miner system into a suite of analysis tools to support experimental analysis to address a problem in structural biology known as biological macromolecular crystallisation [82, 71].

Chapter 7

Case Study: Biological Macromolecular Crystallisation

In this chapter, we describe a scientific problem domain known as *biological macromolecular crystallisation* (§7.1). This domain is data and knowledge rich and is particularly amenable to the application of the supervised learning techniques which we have developed in this thesis. Indeed, this domain was the original motivation for our work. We discuss how Semantic Web technologies are helping to collate and organise data and knowledge in this domain (§7.2) and how DL learning systems like OWL-Miner can be used to mine this data (§7.3) before describing how the OWL-Miner system is currently being integrated into a laboratory setting to aid in experimental analysis (§7.4). Finally, we remark on the current progress of this work, and opportunities for future development (§7.5) before concluding (§7.6).

7.1 Biological Macromolecular Crystallisation

7.1.1 The protein crystallisation bottleneck in structural biology

Recent advancements in proteomics have produced an explosion in the number of proteins as potential drug targets for treating disease. The field of *structural biology* is concerned with ascertaining a precise understanding of the three dimensional structure of such proteins which is crucial in determining their function for use in drug design. Currently, X-ray crystallography is the most accurate and widely used method to determine protein structures. However, the number of protein targets being produced has rapidly outpaced our ability to determine their structure using this method. The necessary step of *protein crystallisation* in the process of X-ray crystallography is currently a major bottleneck, often taking from weeks to years to deliver a protein crystal of sufficient quality for further analysis [17].

Protein crystallisation has an unacceptably high failure rate: studies in structural genomics suggest that, out of around 45,000 soluble, purified target proteins, around 14,000 crystallised and only around 5,000 resulted in a crystal structure [9]. One study [93] reported that, of a set of 96 proteins, it took around 150,000 crystallisation experiments to produce 277 crystal leads for 36 of the proteins; this means that only 0.2% of experiments produced crystal outcomes, and 99.8% produced a different outcome. Such failure rates are commonly encountered for any new protein crystallisation trial.

Protein crystallisation is a very complex physicochemical process with a large number of experimental factors [63]. This results is a vast space of possible experimental conditions, and very little science exists to guide the selection of appropriate conditions for crystallising any given protein *ab initio*. The best methods for protein crystallisation to date are still largely based on random sampling. High-throughput protein crystallisation facilities (such as CSIRO[1] Collaborative Crystallisation Centre, C3[2]) typically tackle the problem using automation, where many crystallisation experiments can be carried out in parallel using robotics. However, such facilities are still failing to keep up with the demand for protein structures and protein crystallisation remains a bottleneck in structural genomics studies. Therefore, methods for assisting crystallographers in making rational choices of experimental conditions which increase the likelihood of generating high quality crystals are sought to increase the efficiency of the protein crystallisation process [86].

7.1.2 The protein crystallisation method

A single protein crystallisation experiment is typically performed by combining a small quantity (often μl) of a purified, homogeneous protein sample with a chemical solution called a crystallant cocktail, or *precipitant*. To coax the protein-protein molecular interactions necessary for their precipitation into a crystalline solid, the experiment aims to increase the concentration of the protein in solution to supersaturation. Figure 7.1 is a *crystallisation phase diagram* [2] describing regions where protein molecules are most likely to interact in solution, increasing in probability from: *stable* where protein molecules are free in solution; *metastable* where short-term protein-protein molecular interactions occur; *labile* where interactions are more fre-

[1]The Commonwealth Scientific and Industrial Research Organisation (CSIRO) is Australia's national science agency. Website: http://www.csiro.au

[2]C3 Website: http://crystal.csiro.au

quent permitting ordered aggregation (crystalline structure formation); to *unstable precipitation* where interactions occur too frequently to form ordered aggregates.

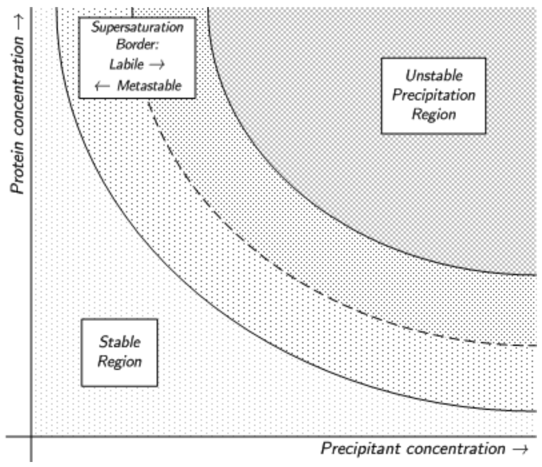


Figure 7.1: A crystallisation phase diagram depicting the regions: stable, metastable, labile and precipitation. Supersaturation starts at the border between stable and metastable and increases in magnitude through the labile region to the precipitation region.

Homogeneous nucleation describes events where protein-protein molecular interactions occur to form small ordered aggregates necessary for crystal growth, and which only occur in the labile region. Once formed, the growth of crystal nuclei are maintained if the system remains in the labile or metastable supersaturated phases. If the system enters the precipitation region, disordered aggregates are formed more rapidly than crystal structures and protein denaturation[3] may occur. On the other hand, if the system enters or never leaves the stable region, crystal growth cannot proceed and any existing crystal structures may dissolve.

The phase of a crystallisation experiment can be observed visually, as depicted in figure 7.2 which labels a number of different experimental states according to their phase. The class described as *clear* often corresponds to the stable or metastable regions; *phase separation* and *crystals* lie within the metastable/labile supersaturated regions; and *precipitate* (and sometimes also *skin*) lie in the unstable precipitation region.

The most popular method to decrease protein solubility in crystallant solution is via the physical exclusion of volatile chemicals from the crystallant. This method, known as *vapor diffusion*, is depicted in figure 7.3.

[3]Denaturation describes an undesirable event where the native structural conformation of a protein is disrupted.

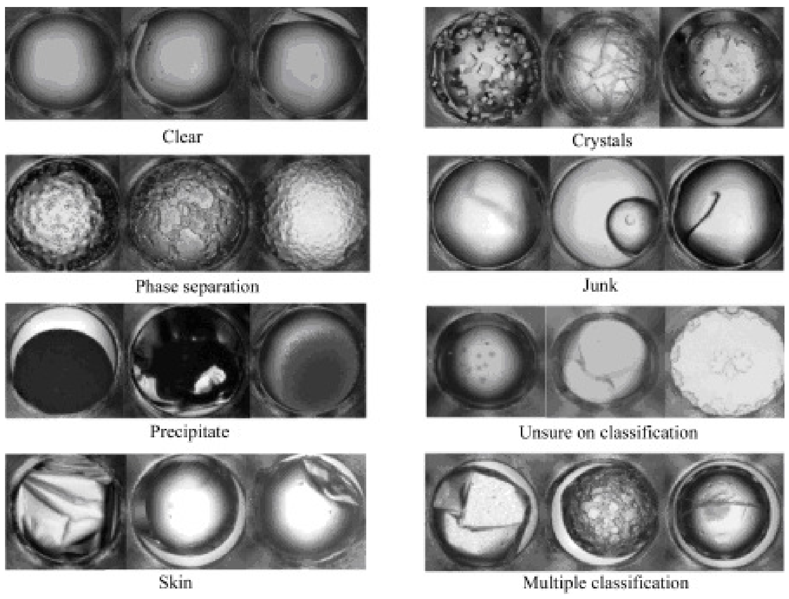


Figure 7.2: Various classifications of protein crystallisation experiment states. This figure shows images of microbatch experiments [93].

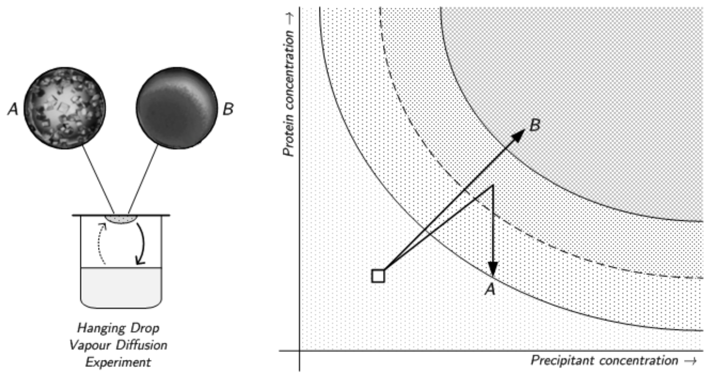


Figure 7.3: A vapour diffusion experiment (by 'hanging drop') and related crystallisation phase diagram. Initially, roughly equal proportions of protein and crystallant solution are suspended in a drop above a reservoir of crystallant solution in a closed chamber. Over time, volatile components of the crystallant (e.g. water) diffuse into the reservoir, decreasing the protein solubility in the drop. If protein molecules nucleate in the labile phase region, they may continue to grow until crystals are formed (path A), else supersaturation of both protein and crystallant may continue into the precipitation phase region potentially resulting in protein denaturation (path B).

The number of different combinations and concentrations of chemicals in crystallant cocktails, kinds of experiments, and methods for decreasing protein solubility available to a crystallographer are infinite. Additionally, there are often no indicators with respect to a protein itself that may suggest a particular course of experiments that will achieve crystallisation. In practice, crystallographers employ screening methods for experimentation broadly based on either *random* or *factorial* designs. In random screening, a range of randomly selected experimental conditions are tested to observe the behaviour of a protein. While not truly random, the popular *sparse matrix screen* is a related strategy where the screen conditions are randomly selected from a range of conditions which have been known to work for a large class of proteins in the past. The goal of such screening is to determine a starting point from which further experiments may be refined (*optimised*) with more screening. In factorial screening, controllable factors are simultaneously and randomly varied in a balanced manner across experiments with an aim to enable statistical analyses of results [16]. In practice, the number of factor levels to vary in a screen can grow very large, so *incomplete* factorial screening is usually performed on a carefully chosen subset of factors. *Grid* screens are also common which fix all but one or two factors at a time.

Given a protein sample to crystallise, a crystallographer will often attempt many hundreds to thousands of screening experiments with no guarantee of success. As principled methods of crystallisation do not yet exist, the choice of experiments in these screens is largely guided by preferences of the individual crystallographer along with their available time and resources.

High-throughput protein crystallisation (HTPC) facilities have emerged in recent years to cope with the increased demand for protein structure information. These facilities often employ robotics to automate the setup and observation of a large number of crystallisation experiments in parallel. One such robotic system is the Rigaku Minstrel HT$^{\text{TM}}$ drop imager[4] which was designed to incubate many experiments by storing plates which can contain from between 24 to 1536 experiments each and to take images of the experiments over time, which are recorded in a database. Crystallographers can then inspect their experiments at any time using a web interface to track and label their state with classifications such as those shown in Figure 7.2.

7.1.3 Mining protein crystallisation data

While data generated throughout the course of a protein crystallisation trial may be used to determine conditions in optimisation experiments, it is common practice amongst most high throughput facilities to ignore or discard such data once a suitable protein crystal has been grown. Only the very final experimental condi-

[4] http://www.rigaku.com/products/automation/minstrelht

tions which yielded a crystal used to produce a structure are reported in scientific publications and public databases such as the Protein Data Bank (PDB) [12] and the Biological Macromolecule Crystallization Database (BMCD) [105].

Data generated in the process of crystallising proteins, particularly data describing the majority of crystallisation attempts perceived by crystallographers to have failed, is a source of valuable information. Machine learning methods could be applied over such data to reveal trends and patterns which, particularly if encoded as human-comprehensible rules, could be used as guiding principles for the rational selection of crystallisation conditions towards quality protein crystals, thus increasing the efficiency of protein crystallisation [36, 74, 86, 94].

Several methods of mining such data have been attempted. The space of experimental parameter values in which a large number of proteins have successfully crystallised has been investigated by several attempts to perform statistical clustering of the BMCD. These methods include Euclidean k-means [31], the results of which were reproduced and extended with an attribute-value rule learning approach [39, 36]. Other mining attempts include clustering over data within individual HTPC facilities which include both positive (crystals grown) and negative (failed experiment) data. Analyses of this data has identified experimental settings which had succeeded in crystallising a large number of proteins, which gave rise to commercially developed ready-made screens, many of which are still in use today [74]. Case-based reasoning [45] and neural networks [26] have been employed as predictive models in this setting to provide decision support for crystallographers in selecting experimental conditions which may be likely to crystallise proteins.

In terms of data mining approaches which are capable of generating comprehensible rules to aid in human understanding, the most useful methods to date for clustering and prediction over protein crystallisation data appear to be attribute-value rule-based approaches [45, 39, 36, 21]. Neural networks models, despite their accuracy, are incomprehensible and cannot be interpreted to reveal knowledge which can aid in understanding how to crystallise proteins and to translate into actions for crystallographers. Studies on mining attribute-value rules from data in the BMCD [39, 36] report that augmenting the empirical data with hierarchical background knowledge (such as chemical species and their properties) improved the comprehensibility and expressive power of the hypotheses generated, as well as providing new features for learning. However, the expressiveness of the propositional attribute-value rule-based hypotheses limits such approaches from capturing complex multi-parameter associations which are expected to be present in crystallisation data [94]. These studies have had little practical impact because of the general nature of the knowledge discovered and the fact that statistical analyses of the BMCD are fundamentally ineffective as they do not take into consideration the number of trials or failed crystallisation attempts [86].

7.1.4 Integrating heterogeneous protein crystallisation data

Efforts to integrate protein crystallisation experimental data across HTPC facilities for large-scale analyses like data mining have been hampered by the heterogeneity of the data and inconsistency in the naming of entities referred to in experiments, such as chemical names [75]. In order to overcome these difficulties, the Crystallisation Data Exchange (XDX) Ontology Consortium[5] was established in 2011 by CSIRO and several of the world's largest leading HTPC facilities to develop an ontology with the Web Ontology Language (OWL)[6] to comprehensively describe the semantics of, and a standard nomenclature for, protein crystallisation experimental data [71] which is required to address the data integration problem [28]. The XDX ontology incorporates specific domain knowledge in protein crystallisation as well as published sources of relevant knowledge such as the ChEBI OWL ontology for chemistry [25] to capture chemical identifiers and relationships, and the Protein Ontology [70] for capturing protein features. Efforts are currently underway in CSIRO to map heterogeneous crystallisation experimental data from various HTPC facilities using the Resource Description Framework (RDF) data model[7]. This effort is aligned with the vision of the Semantic Web [90] whereby RDF is used to capture and publish data in a machine interpretable way, and which may be directly published and associated to other data sets as part of the Linked Open Data (LOD) project [13] for wider scrutiny [82].

The push to formally capture the semantics of protein crystallisation data and knowledge for data integration with formal ontologies makes protein crystallisation an ideal domain to explore the development of knowledge-intensive machine learning and data mining technologies directly over Semantic Web formalisms, and is the primary motivation for the work of this thesis towards the development of the OWL-Miner system [80]. In the next section, we will describe how data and knowledge in

[5]http://www.xdx-ontology.org

[6]http://www.w3.org/TR/owl-overview/

[7]http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

this domain is captured, and how OWL-Miner is being used to analyse the data to support crystallographers with their experimental analysis tasks.

7.2 Description Logic learning for protein crystallisation

Motivated by the use of Semantic Web technologies by the protein crystallisation community for capturing experimental data and knowledge, we are currently working with the CSIRO C3 laboratory to install the OWL-Miner system to aid crystallographers with data analysis tasks. In this section, we will describe how experimental data are captured under the XDX OWL ontology, how experimental data is enriched with knowledge from other ontologies such as ChEBI OWL, and the application of OWL-Miner to perform supervised learning over resultant combined knowledge bases. We will then describe how these tasks can aid the crystallographers in designing their experiments, and remark on how this technique can be applied to larger collections of crystallography data in general.

7.2.1 Experimental Data and Knowledge Representation

In our work, protein crystallisation experiments are captured in RDF as data assertions against concept and role names from the XDX OWL ontology. To illustrate an example of an individual experiment, Figure 7.4 presents a set of assertions to describe how the features of the experiment are captured against components of the XDX OWL ontology.

Figure 7.4 partially describes a set of assertions about an individual experiment consisting of a single drop in a hanging vapour diffusion experiment at 20° centigrade, containing 0.2M of magnesium acetate at pH 6.0, amongst other chemicals. The experiment is attributed with two observations made on different days, one where the observed state of the drop was clear, and another five days later indicating the presence of crystalline structures.

The structure of experiments is amenable for representation by relational structures such as DLs, as each experiment may contain different sets of chemicals each with their own properties from a set of several thousand chemicals. Furthermore, supplementary data and knowledge exists outside of the domain of protein crystallisation which can be added to enrich the experimental descriptions. One such data source is the ChEBI OWL ontology which contains hierarchical and compositional knowledge about chemical species which cover those used in crystallisation. For

$$\begin{array}{lll}
e_0 : \mathit{Drop} & \langle e_0, \texttt{double}[\mathit{20.0}] \rangle : \mathit{atTempC} & \\
 & \langle e_0, m \rangle : \mathit{method} & m : \mathit{HangingDropVaporDiffusion} \\
 & \langle e_0, c_0 \rangle : \mathit{hasComponent} & c_0 : \mathit{Component} \\
 & \langle c_0, s_0 \rangle : \mathit{hasConcentration} & s_0 : \mathit{Concentration} \\
 & \langle s_0, M \rangle : \mathit{unitOfMeasure} & \\
 & \langle s_0, \texttt{double}[\mathit{0.2}] \rangle : \mathit{hasValue} & \\
 & \langle c_0, \mathit{mgac} \rangle : \mathit{hasChemical} & \mathit{mgac} : \mathit{Chemical} \\
 & \langle c_0, \texttt{double}[\mathit{6.0}] \rangle : \mathit{pH} & \\
 & \ldots & \\
 & \langle e_0, o_0 \rangle : \mathit{hasObservation} & o_0 : \mathit{Observation} \\
 & \langle o_0, \mathit{clear} \rangle : \mathit{observedState} & \mathit{clear} : \mathit{State} \\
 & \langle o_0, \texttt{date}[\mathit{2014\text{-}12\text{-}14}] \rangle : \mathit{atTime} & \\
 & \langle e_0, o_1 \rangle : \mathit{hasObservation} & o_0 : \mathit{Observation} \\
 & \langle o_1, \mathit{precipitate} \rangle : \mathit{observedState} & \mathit{precipitate} : \mathit{State} \\
 & \langle o_1, \texttt{date}[\mathit{2014\text{-}12\text{-}19}] \rangle : \mathit{atTime} &
\end{array}$$

Figure 7.4: A description of an individual protein crystallisation experiment expressed as assertions over concepts and roles from the XDX OWL ontology.

example, knowledge which can be added about magnesium acetate from ChEBI is shown below in Figure 7.5.

$$\begin{array}{l}
\mathit{mgac} : \mathit{CHEBI\_62964} \\
\mathit{CHEBI\_62964} \sqsubseteq \exists \mathit{hasFunctionalParent}.(\mathit{AceticAcid}) \\
\mathit{CHEBI\_62964} \sqsubseteq \exists \mathit{hasPart}.(\mathit{MagnesiumAtom}) \\
\mathit{CHEBI\_62964} \sqsubseteq \exists \mathit{hasPart}.(\mathit{Cation}) \\
\mathit{CHEBI\_62964} \sqsubseteq \exists \mathit{hasPart}.(\mathit{Anion}) \\
\mathit{CHEBI\_62964} \sqsubseteq \exists \mathit{hasMolecularWeight}.(\texttt{double}[= 142.393])
\end{array}$$

Figure 7.5: Knowledge and data about the chemical compound *magnesium acetate* from the ChEBI OWL ontology. These axioms describe how instances of *CHEBI_62964* refer to magnesium acetate which have part magnesium atom, a cation and an anion (meaning the compound is a salt), is structurally related to acetic acid, and that it has an average molecular weight of 142.393 g/mol.

The ChEBI ontology provides an identification scheme for chemicals and organises them into a classification hierarchy based on structural features, properties and roles [38]. Adding extra information such as axioms from the ChEBI ontology to a knowledge base containing protein crystallisation experiments provides new features which can be leveraged by supervised learning systems such as OWL-Miner to learn concepts for classifying groups of experiments based on such features. For example, the ability to classify a collection of experiments which contain chemicals of a certain class, or which contain a functional structure or part, might be an important predictor of a response like experimental failure. Extra quantitative data such as the molecular weight of chemical species or their pH values can also be used in this way. In the next section, we will describe how knowledge bases consisting of this information can be used in conjunction with learning systems like OWL-Miner to perform classification and subgroup discovery to aid crystallographers interpret the results of their experiments.

7.3 Data Mining Protein Crystallisation Conditions

7.3.1 Single Target Experimentation

Given a new protein target to analyse, a crystallographer will often perform a number of experiments with a so-called *random screening* approach which samples multiple chemicals and conditions in a sparse manner. The purpose of these initial screens is to ascertain the behaviour of the protein in a number of conditions and to locate leads for further analysis, depending on the response of the protein to each condition. Such screens can often consist of tens to hundreds of experiments which sample many different chemicals and conditions. Interpreting the space of parameters involved can be a manual and time-consuming process [63]. Manual interpretation of the results of such screens is usually performed in such a way that the best individual results are singled out for fine-grained screening, such as incomplete factorial screening around a very restricted set of chemicals and conditions, or grid screening which focuses on a fixed set of chemicals and conditions while varying a small number of parameters such as concentration or pH levels. This approach necessarily requires the protein under consideration to have produced reasonable outcomes after initial screening to permit further experimentation to proceed towards narrower sets of chemicals and conditions. However, it is often the case that a protein will not crystallise after initial random screens. In this case, the variety of responses must be manually interpreted by the crystallographer who will then make an educated guess as to which experiments to try next. When the responses are primarily negative, such as where most experiments result in clear drops or drops containing precipitate, it is often unclear which experiments are the best ones to try next amongst a large set of diverse experiments.

To aid in the interpretation of the results of screening, several software packages have been developed (e.g., AutoSherlock [69]) which visualise the experimental results, but do not provide the ability to perform automated clustering of experiments per result based on phase states. If computed automatically, such clustering has the potential to provide valuable and rapid insight into complex multi-parameter combinations affecting crystallisation for individual proteins [94]. By clustering in this way, a crystallographer can determine which experimental parameters are most highly correlated with certain outcomes which can inform how to proceed with further experimentation, even when crystalline responses are not achieved. For example, experiments which transition quickly from clear drops to precipitate are interpreted as having conditions which force a protein sample directly into the unstable region of phase space (Figure 7.3), a negative outcome. Understanding the primary factors of the experiments which produce this result can aid a crystallographer in knowing which chemicals and their concentrations or pH levels to avoid. Alternatively, experiments which transition relatively slowly from clear to precipitate and are not associated with other negative outcomes such as skin or phase separation (Figure 7.2) may be interpreted as having passed from the stable region of phase space through labile and metastable where crystal growth ought to occur and into the unstable region. Such behaviour can be interpreted as positive, as it suggests the conditions for the protein are nearly correct, and deserve further attention even though no crystalline responses were produced. In this way, a crystallographer can make use of data mining techniques to describe clusters of experiments over their features such as their constituent chemicals and conditions which most highly correlate with known behaviours which are interpreted to be helpful or a hindrance to crystallogenesis. By clearly elucidating these factors over a number of experiments, the crystallographer may use this information to inform how further experiments should be carried out.

The descriptions of protein crystallisation experiments as shown in Section 7.2 are ideally suited for processing by the OWL-Miner system which was developed primarily for the purpose of interpreting results such as these. Given an initial screen of experiments which represent a sparse sampling of the space of possible parameters around chemical and conditions, OWL-Miner can be used to perform supervised subgroup discovery over the experiments to determine which of the features most highly correlate with certain responses. Subgroup discovery is ideally suited to this task, as there may be isolated pockets of chemicals and conditions in the space of parameters which correlate highly with certain responses but which may not cover the full dataset. The hypotheses produced to describe subgroups which correlate most highly with positive outcomes can then be used to guide further experimentation, with the crystallographer focussing on fine-grained incomplete factorial or grid sampling which involve chemicals matching the descriptions. In Section 7.4, we present a real example where this kind of analysis was performed in retrospect over a fully annotated and completed experiment for a single protein called *orotic acid hydrolase* (§7.4.2).

As experimentation progresses towards sampling smaller areas of the parameter space with a reduced number of chemicals and conditions, OWL-Miner can be also used to perform supervised classification to infer hypotheses which are intended to range over all experiments with certain responses. The use of DL learning with systems like OWL-Miner are clearly amenable to these tasks as the results are comprehensible models which describe the parameter space directly in terms of chemicals and conditions which the crystallographers can use to understand how further experiments should be designed.

7.3.2 Multiple Target Interpretation

Given a large collection of experiments accounting for the conditions and responses for a number of proteins, patterns in the global space of parameters can be investigated to determine which conditions are most strongly correlated with particular responses. Indeed, this is precisely how most modern pre-manufactured crystallisation screening kits are produced, which is to analyse the most commonly successful chemical conditions which have succeeded in crystallising a large number of proteins in the past [74]. While such work has been undertaken over empirical data in the BMCD [36], these data represent only positive outcomes extracted from publications which describe the final successful conditions used to achieve crystallogenesis for each contributed protein, and ignore all other results such as intermediate behaviours or negative responses. Furthermore, the learning over this dataset did not incorporate more general knowledge about the domain, such as that which is otherwise provided by ChEBI around chemical classifications, structures and functions. We anticipate that by incorporating such extra knowledge, that cluster descriptions of experimental conditions over such terms may reveal properties which are influencing crystallogenesis which have not been discovered before.

It is the ultimate goal of the XDX OWL Ontology Consortium to provide a standard nomenclature for capturing experimental conditions in a way in which such data and knowledge can be integrated from a number of contributing sources as a step toward this goal. The OWL-Miner system was designed to support analysis directly over such a data and knowledge holding for mining commonly successful or unsuccessful conditions. Efforts are currently still ongoing to further develop the XDX OWL ontology and supporting tools to facilitate their use by HTPC centres which hold valuable data in heterogeneous formats and storage mechanisms which aren't otherwise easily shared.

7.4 The CSIRO Collaborative Crystallisation Centre (C3)

The CSIRO Collaborative Crystallisation Centre (C3) in Parkville, Melbourne is a world leading protein crystallisation research facility. This centre is host to HTPC machinery for automating the setup and observation of many crystallisation experiments, and maintains a rigorous database of experimental descriptions and their outcomes. We have partnered with the director of C3, Dr. Janet Newman, to investigate the application of the OWL-Miner system to experimental data held by C3. To support this task, a large amount of supporting software and infrastructure has been put in place which we refer to as the X-plorer system.

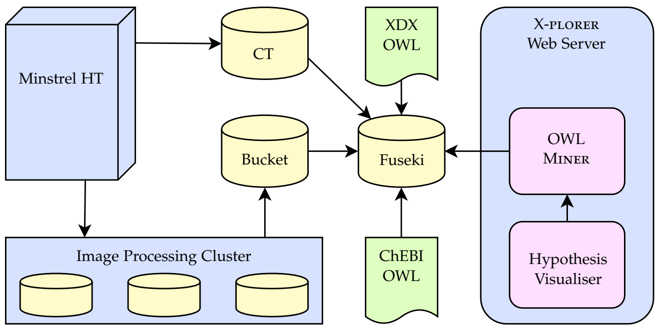


Figure 7.6: The architectural ecosystem of the infrastructure supporting the X-plorer system for the C3 laboratory.

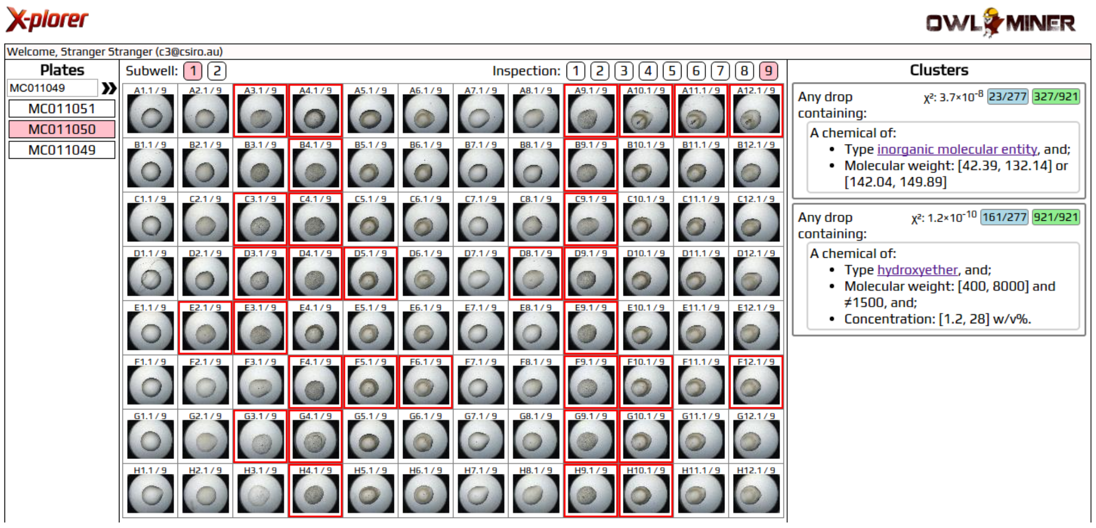
Figure 7.6 shows a high-level architectural diagram outlining the major software and hardware components which have been put in place to support the use of OWL-Miner via the X-plorer system. Automated experiment incubation and monitoring hardware known as the Rigaku Minstrel HT^{TM} system accepts plates containing multiple crystallisation experiments and uses a camera to take regular images of each experiment. These images are then stored in a database which the crystallographers can inspect with specialised software called CrystalTrak^{TM}. This software permits crystallographers to view and label their experiments at each inspection time with their interpretation of each response, such as clear, precipitate, skin, and so on. This software also supports the design of new experimental screens based on manually entered sets of parameters chosen by the crystallographers for further experimentation, but necessarily requires that the crystallographers manually interpret the results and select the space of parameters for further testing, which can be a complex and time-consuming process.

To automate the process of analysing the response of the many experiments by their images which are produced by the Minstrel system, generated images are placed into distributed storage on a compute cluster. On this cluster are several computer vision classifiers which have been trained to recognise certain features in drop images [108, 107], and an ensemble learner is used to automatically attribute labels to each image depending on the response identified. Automated labels are then captured in a separate database labelled as 'Bucket' in Figure 7.6. By providing automated scoring for images, the crystallographers are free to focus on the task of analysing the parameter space relative to the various responses. To support the latter task, the X-PLORER system implements procedures for extracting data and knowledge from both the 'CT' and 'Bucket' databases which respectively contain the experimental descriptions and their responses. Once extracted, this data is transformed to RDF under the XDX OWL ontology and stored in a Fuseki[8] triplestore database in preparation for analysis by OWL-MINER.

The X-PLORER system also maintains mappings between the chemical entities referred to in experiments in the 'CT' database and standard terms and synonyms that are used in other data sets, specifically ChEBI and PubChem. Using this data, inclusion axioms which classify chemicals and specify functional parts amongst other knowledge is extracted from the ChEBI OWL ontology and added to the Fuseki database of experimental descriptions, such as those shown in Figure 7.5. Furthermore, any extra or missing quantitative data about chemicals is added from PubChem, which also serves to cross-reference values against existing data to indicate possible errors, for example, where there is disagreement about the molecular weight of a chemical substance. The process of extracting knowledge from the ChEBI ontology proceeds by identifying the concepts for which the chemicals used in experiments are instances, and extracting all inclusion axioms which refer to these classes up the classified subsumption hierarchy recursively until all relevant axioms are extracted. The collective expressivity of the axioms extracted from ChEBI OWL are selected so as to not exceed $\mathcal{EL}$, as the XDX OWL ontology is also expressed in $\mathcal{EL}$, resulting in a knowledge base for which classification is efficient to compute.

[8]Apache Jena Fuseki: https://jena.apache.org/documentation/fuseki2/index.html

Once entirely transformed to RDF under OWL, the experimental data from the 'CT' database and their combined automated and manual response labelling from the 'Bucket' database can then be analysed directly with OWL-Miner. The X-plorer system supports the use of OWL-Miner in a number of ways which greatly simplify user interaction.


X-plorer
OWL MINER
Welcome, Stranger Stranger (c3@csiro.au)
Plates
MC011049
MC011051
MC011050
MC011049
Subwell: 1 2
Inspection: 1 2 3 4 5 6 7 8 9
A1.1 / 9 A2.1 / 9 A3.1 / 9 A4.1 / 9 A5.1 / 9 A6.1 / 9 A7.1 / 9 A8.1 / 9 A9.1 / 9 A10.1 / 9 A11.1 / 9 A12.1 / 9
B1.1 / 9 B2.1 / 9 B3.1 / 9 B4.1 / 9 B5.1 / 9 B6.1 / 9 B7.1 / 9 B8.1 / 9 B9.1 / 9 B10.1 / 9 B11.1 / 9 B12.1 / 9
C1.1 / 9 C2.1 / 9 C3.1 / 9 C4.1 / 9 C5.1 / 9 C6.1 / 9 C7.1 / 9 C8.1 / 9 C9.1 / 9 C10.1 / 9 C11.1 / 9 C12.1 / 9
D1.1 / 9 D2.1 / 9 D3.1 / 9 D4.1 / 9 D5.1 / 9 D6.1 / 9 D7.1 / 9 D8.1 / 9 D9.1 / 9 D10.1 / 9 D11.1 / 9 D12.1 / 9
E1.1 / 9 E2.1 / 9 E3.1 / 9 E4.1 / 9 E5.1 / 9 E6.1 / 9 E7.1 / 9 E8.1 / 9 E9.1 / 9 E10.1 / 9 E11.1 / 9 E12.1 / 9
F1.1 / 9 F2.1 / 9 F3.1 / 9 F4.1 / 9 F5.1 / 9 F6.1 / 9 F7.1 / 9 F8.1 / 9 F9.1 / 9 F10.1 / 9 F11.1 / 9 F12.1 / 9
G1.1 / 9 G2.1 / 9 G3.1 / 9 G4.1 / 9 G5.1 / 9 G6.1 / 9 G7.1 / 9 G8.1 / 9 G9.1 / 9 G10.1 / 9 G11.1 / 9 G12.1 / 9
H1.1 / 9 H2.1 / 9 H3.1 / 9 H4.1 / 9 H5.1 / 9 H6.1 / 9 H7.1 / 9 H8.1 / 9 H9.1 / 9 H10.1 / 9 H11.1 / 9 H12.1 / 9
Clusters

Any drop containing: χ^2: 3.7×10^{-8} 23/277 327/921

A chemical of:
- Type inorganic molecular entity, and;
- Molecular weight: [42.39, 132.14] or [142.04, 149.89]

Any drop containing: χ^2: 1.2×10^{-10} 161/277 921/921

A chemical of:
- Type hydroxyether, and;
- Molecular weight: [400, 8000] and ≠1500, and;
- Concentration: [1.2, 28] w/v%.

Figure 7.7: A screenshot of the X-plorer system which allows crystallographers to visualise their experiments, select training sets either manually or based on experimental responses, and to request subgroup descriptions from the OWL-Miner system. While OWL-Miner produces DL concepts as hypotheses, the X-plorer system translates these into structured natural language. Each subgroup description is also attributed with the strength of the score used, in this case the value for the χ^2 measure, and each are also paired with the set of experiments which are covered to permit their visualisation in the main screen.

To use the X-plorer system, a crystallographer logs in via http://crystallisation.csiro.au (not publicly accessible) and enters a number of barcodes corresponding to plates containing their experiments. The system then loads a visualisation of these experiments by plate onto the screen as shown in Figure 7.7. The software permits the user to inspect the specific conditions of each experiment and can highlight them based on automatically or manually labelled responses. By selecting a set of experiments from a number of plates, the user can then select a positive and negative training set for supervised learning based on a manual selection, or based on a common set of responses. For example, the user can select a collection of drops with crystalline and precipitant responses as the set of positives, and clear drops as the set of negatives. Once a training set is identified, the user can select parameters for

OWL-Miner to begin learning hypotheses, from the problem to be solved (classification or subgroup discovery), the measure to be used (χ^2, weighted relative accuracy, etc.), minimum thresholds on the measure, and maximum limits on the number of hypotheses to locate and time spent searching for them.

Of the options that can be selected, OWL-Miner also supports the explicit inclusion or exclusion of ontology terms to use in the generation of hypotheses. It is planned to permit the user of X-plorer to make such selections on top of a core set of features such as chemical species which form the basis of terms with which OWL-Miner will attempt to construct hypotheses, for example, only pH levels. In this way, the user is free to use OWL-Miner to explore whether there are such concepts which have strong correlations to experiments with certain responses to test hypotheses. Otherwise, the OWL-Miner system can be used without exclusions and can be used to derive any and all hypotheses which meet the problem specification provided by the user.

Once OWL-Miner begins processing over a set of labelled experiments, the current set of best hypotheses are returned to the X-plorer interface on a continual basis to be visualised on the web-page until OWL-Miner terminates. However, the raw DL concept descriptions are not used directly, as the intended users are potentially crystallographers with no knowledge of DLs. While a DL concept expression may be easily understood with minimal training, the X-plorer system implements a novel translation mechanism which maps DL concepts to structured natural language expressions as can be seen in Figure 7.7. These expressions are nested and are tightly linked to the DL concepts returned, and are attributed with links to source information, such as the link to reveal more information about *hydroxyether* compounds which takes the user to the ChEBI website. The method used is based on templating, which is to recognise certain DL concept fragments and to map these to structured English representations for rendering as HTML. For example, the concept:

$$
\begin{aligned}
&\exists \mathit{hasComponent}.(\mathit{Component} \sqcap \\
&\quad \exists \mathit{hasChemical}.(\mathit{Chemical} \sqcap \mathit{CHEBI\_46789} \sqcap \\
&\quad\quad \exists \mathit{hasMolecularWeight}.(\texttt{double}[(\geq 400 \wedge < 1500) \vee (> 1500 \wedge \leq 8000)])) \sqcap \\
&\quad \exists \mathit{hasConcentration}.(\mathit{Concentration} \sqcap \\
&\quad\quad \exists \mathit{hasValue}.(\texttt{double}[\geq 1.2 \wedge \leq 28]) \sqcap \\
&\quad\quad \exists \mathit{unitOfMeasure}.(\{w/v\%\}))
\end{aligned}
$$

is simplified, condensed and represented textually as:

A chemical of:

- Type: hydroxyether, and;
- Molecular weight: [400, 8000] and $\neq$ 1500, and;
- Concentration: [1.2, 28] w/v%.

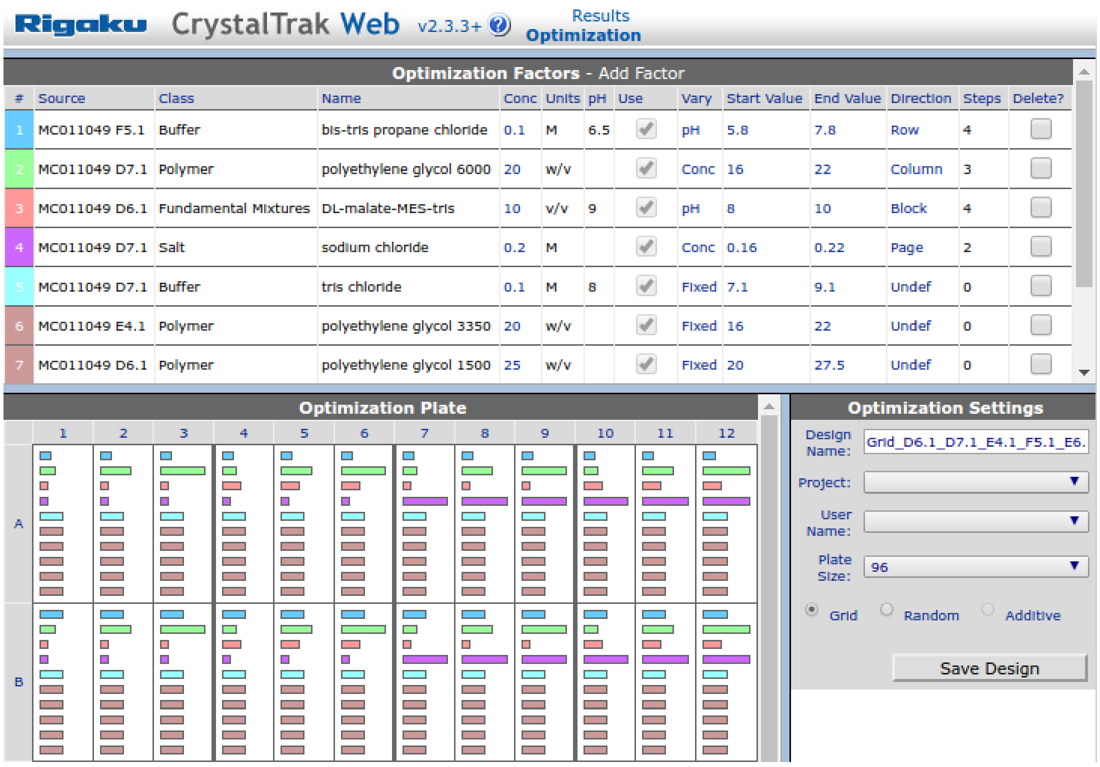
the latter clearly based directly on the concept itself, however is more readable. Note the replacement of the concept *CHEBI_46789* with its label 'hydroxyether'. The implementation of this translation method is problem-specific, as it is based on hard-coded rules around the kinds of DL concept fragments which are expected to be produced by OWL-Miner over the XDX OWL and ChEBI OWL terminologies.

When a user highlights a particular concept found by OWL-Miner in the X-plorer interface, the set of experiments which are covered by the concept are highlighted, along with the classification designation of each experiment relative to the concept, namely all true and false positives, and all true and false negatives. This provides the user with a quick visual guide as to the appearance of which experiments were covered by the concept or not relative to the nominated positive and negative training set.

7.4.1 Optimisation Screen Design

The role of OWL-Miner in the X-plorer system is to aid the crystallographer in identifying the factors which are significantly correlated with particular experimental responses. Once identified, the crystallographer may then wish to create new experimental screens around the conditions they have identified as being suitable for further analysis. This step is known as optimisation, as given a set of experiments which did not produce a usable protein crystal structure, parameters are selected for modification towards optimising the protein conditions towards the growth of high-quality crystal structures. The existing CrystalTrak^{TM} software supports the creation of new experiments by plate designs around the selection of one or more existing experiments by creating new grid, random, or additive design methods across a set of selected parameters.

By selecting a number of experiments which were covered by a concept produced by OWL-Miner as being strongly correlated with particular outcomes, the features described by the concept give an indication of which parameters should be varied in the new screen. For example, if pH was identified as a feature in a concept which



Optimization Factors - Add Factor

#	Source	Class	Name	Conc	Units	pH	Use	Vary	Start Value	End Value	Direction	Steps	Delete?
1	MC011049 F5.1	Buffer	bis-tris propane chloride	0.1	M	6.5	✓	pH	5.8	7.8	Row	4	☐
2	MC011049 D7.1	Polymer	polyethylene glycol 6000	20	w/v		✓	Conc	16	22	Column	3	☐
3	MC011049 D6.1	Fundamental Mixtures	DL-malate-MES-tris	10	v/v	9	✓	pH	8	10	Block	4	☐
4	MC011049 D7.1	Salt	sodium chloride	0.2	M		✓	Conc	0.16	0.22	Page	2	☐
5	MC011049 D7.1	Buffer	tris chloride	0.1	M	8	✓	Fixed	7.1	9.1	Undef	0	☐
6	MC011049 E4.1	Polymer	polyethylene glycol 3350	20	w/v		✓	Fixed	16	22	Undef	0	☐
7	MC011049 D6.1	Polymer	polyethylene glycol 1500	25	w/v		✓	Fixed	20	27.5	Undef	0	☐

Optimization Settings

Design Name:	Grid_D6.1_D7.1_E4.1_F5.1_E6.
Project:	
User Name:	
Plate Size:	96

[✓ Grid] Random Additive

Figure 7.8: A screenshot of the Rigaku CrystalTrak^{TM} system for creating optimisation screens based on a number of experimental parameters.

was highly correlated with a positive response such as crystals or suitable precipitate, this can be selected for variation over the range as suggested by the constraint on this feature in the concept. Furthermore, if OWL-Miner produced a concept which was correlated highly with negative responses such as unsuitable precipitate or skin, these rules can be combined in the screen design process to ensure that the newly sampled space does not overlap. Designing screens by integrating knowledge determined by OWL-Miner in this way supports crystallographers in more closely and explicitly identifying areas of parameter space which are suitable for further analysis, and is a capability which is not offered by CrystalTrak^{TM}, Formulatrix RockMaker^{TM}[9], or any other crystallisation design tool known to the author.

[9]Formulatrix RockMaker^{TM}: http://formulatrix.com/protein-crystallization/products/rock-maker/index.html

7.4.2 Orotic Acid Hydrolase

A recent project conducted at the C3 sought to determine the structure of a biological macromolecule called *orotic acid hydrolase* (OAH)[10]. Samples of OAH were used in a number of crystallisation experiments, 768 in initial random screening and a further 1,920 in optimisation screening. Notably, every experiment in this project was manually annotated by an expert crystallographer so as to provide labels to support automated supervised learning.

During the initial random screening phase, 19 of the 768 experiments produced responses which were favourable towards crystallisation, where the remaining 749 experiments were not considered interesting for further analysis. In order to determine the primary factors which were correlated with near-crystalline responses over others, we ran an experiment with OWL-Miner to perform supervised subgroup discovery with the χ^2 measure with a p-value thresholded at 10^{-6}. The initial population consisted of 19 near-crystalline experiments labelled as positives (P), and 749 non-crystalline experiments labelled as negatives (N). OWL-Miner quickly produced several simple hypotheses which are listed in Table 7.1.

No.	**Hypothesis**	**Cover**	χ^2 p-value
1.	$Drop \sqcap \exists hasComponent.(Component \sqcap \exists hasChemical.(CHEBI\_62964))$	P: 5/19 N: 3/749	1.0×10^{-20}
2.	$Drop \sqcap \exists hasComponent.(Component \sqcap \exists hasChemical.(CHEBI\_36364))$	P: 13/19 N: 63/749	2.2×10^{-16}
3.	$Drop \sqcap \exists hasComponent.(Component \sqcap \exists hasChemical.(CHEBI\_33975))$	P: 9/19 N: 47/749	5.7×10^{-11}
4.	$Drop \sqcap \exists hasComponent.(Component \sqcap \exists hasChemical.(CHEBI\_35156))$	P: 4/19 N: 16/749	4.5×10^{-7}
5.	$Drop \sqcap \exists hasComponent.(Component \sqcap \exists hasChemical.(CHEBI\_23114 \sqcap \neg CHEBI\_26710 \sqcap \neg CHEBI\_31206 \sqcap \exists hasMolWt.(\texttt{double}[42.39, 136.28]) \sqcap \exists pH.(\texttt{double}[0.0, 7.5])))$	P: 8/19 N: 52/749	6.1×10^{-8}

Table 7.1: Hypotheses generated by OWL-Miner to describe the factors strongly correlated to experiments with crystalline responses for orotic acid hydrolase (OAH).

The hypotheses listed in Table 7.1 describe to the crystallographer that experiments

[10]Orotic acid hydrolase: https://www.ebi.ac.uk/pdbe/entry/pdb/5hy0

containing:

1. Magnesium acetate (*CHEBI_62964*) tended to crystallise more often than not (5:3);
2. Experiments containing any chemical belonging to the class of alkaline earth salts (*CHEBI_36364*), which includes magnesium and calcium salts, tended to be correlated more strongly with crystalline responses (13/19, 68.4%) than non-crystalline responses (63/749, 8.4%).
3. Magnesium salts (*CHEBI_33975*) and calcium salts (*CHEBI_35156*) were correlated more strongly with crystalline responses (magensium: 9/19, 47.4%; calcium: 4/19, 21%) than non-crystalline responses (magnesium: 47/749, 6.2%; calcium: 16/749, 2.1%).
4. Chloride salts (*CHEBI_23114*) having a molecular weight between $[42.39, 136.28]$ and a pH between $[0.0, 7.5]$ except specifically sodium chloride (*CHEBI_26710*) and ammonium chloride (*CHEBI_31206*) were correlated more strongly with crystalline responses (8/19, 42.1%) than non-crystalline responses (52/749, 6.9%).

These results clearly indicate that specific experimental features such as certain chemical species (i.e., magnesium, calcium and chloride salts) and particular molecular weight and pH ranges were significant to achieve near-crystalline conditions. In this project, the expert crystallographer chose to proceed after initial random screening with fine-grained optimisation sampling around variations of experiments containing magnesium and calcium salts after visual inspection of the results. Ultimately, crystals of sufficient quality for analysis were produced in this phase which resolved the structure of OAH with experimental conditions which included magnesium acetate.

Interestingly, the last hypothesis in Table 7.1 indicated that a number of chloride salts were strongly correlated with near-crystalline responses, however these were not explicitly chosen by the crystallographer for analysis in optimisation screening. However, as magnesium salts were chosen for optimisation, magnesium chloride was included (which is a magnesium, chloride and alkaline earth salt). Post-hoc analysis of the optimisation screen results revealed that experimental conditions which included magnesium chloride correlated most strongly with crystalline conditions over any other condition (116 crystalline responses our of a total of 580 containing magnesium chloride, or 20%), whereas conditions with calcium salts did not perform nearly as well (8 crystalline responses out of a total of 309 containing any calcium salts, or 0.03%). The crystallographers conceded that if they could have determined that chloride salts were strongly correlated with favourable results as is shown by OWL-Miner over initial screening, that these chemicals should have been included in more optimisation experiments. The identification of influential classes of chemicals through the use of the CheBI OWL ontology, along with constraints defining specific sets of chemicals over properties in the XDX OWL ontology, is seen as a powerful mechanism by which to describe possible areas of the space of chemicals which can be focused on for experimentation which permit the crystallographer to make more informed choices.

7.5 Current and Future Work

Currently, the X-plorer system is under active development. The steps which take generated images for automated classification on the CSIRO compute cluster have only recently been implemented, and the labelling data are being used to attribute particular responses to each experiment based on sequences of their observations, such as transitioning from clear to precipitate. Furthermore, the step which takes the set of experiments identified as being covered by a DL concept by OWL-Miner for use in optimisation screen design is incomplete. Shortly, crystallographers working at C3 will be able to take the concepts generated by OWL-Miner and use them directly to interpret per-protein results and to inform the construction of optimisation screens. Once this is achieved, crystallisation experiments which are set up on the basis of this method can be evaluated against past success rates based on manual processing to determine if the efficiency of experimentation has improved, as we reasonably anticipate.

An interesting avenue for future work in this space is to apply OWL-Miner over the ever-growing RDF dataset of experiments in the Fuseki database. With very many experiments for a variety of proteins covering a range of responses considered both negative and positive, OWL-Miner can be used to produce concepts which correlate highly with success or failure across many different proteins as discussed in Section 7.3.2. Performing this analysis may reveal experimental parameters which are interesting for protein crystallography generally, such as revealing correlations between the use of sets of parameter settings and certain responses. As this is the motivating goal of the XDX Ontology Consortium, performing this step to produce interesting results on a single laboratory basis is a worthwhile step towards persuading other laboratories to adopt our approach, and to potentially combine their experimental data which has been standardised under the ontology for a broader data mining effort.

7.6 Conclusion

This chapter has explored the application of the DL learning techniques developed in this thesis and implemented as OWL-Miner in a real-world setting. Indeed, the development of the novel methods in this thesis and the implementation of OWL-Miner was done to support this domain specifically. While work in this area is still progressing, the approach holds great promise for improving the efficiency and effectiveness of experimentation in protein crystallography. This work has become particularly important of late, not only because of the inherent inefficiency of protein crystallisation given the high demand for protein structures, but because the emerging method of cryo-electron microscopy [52] is capable of producing structures for large biological molecules in a fraction of the time typically taken by protein crystallisation methods. As the analysis of small biological targets is currently only possible with X-ray crystallography, crystallographers are even more motivated to improve the rate at which protein crystallisation can be performed in order to keep pace with these new methods.

Conclusions and Future Work

We conclude this thesis with a summary of our research contributions (§8.1) and directions for future work (§8.2).

8.1 Thesis Contributions

In this thesis, we investigated the topic of efficient concept learning over large knowledge bases with highly expressive DL languages, such as $\mathcal{SROIQ}(\mathcal{D})$ which underpins OWL2-DL. In particular, our goal was to improve on the performance and scalability of modern DL learning tools so as to improve their applicability for solving a variety of machine learning and data mining problems. To this end, our novel contributions are:

- The formalisation of the problem of learning from closed-world interpretation in DLs (§3.5, Definition3.5.1) and the definition of a novel context-specific closed-world interpretation suitable for DL learning (§4.2.1, Definition 4.2.8) [80];

- Practical methods for DL learning that use a potentially infinite context-specific closed-world interpretation in the construction and use of a *context graph* (§4.2.2, Definition 4.2.12);

- The formalisation of downward ($\rho_{\bar{\lambda}}$) and upward ($\upsilon_{\bar{\lambda}}$) context-specific refinement operators which are designed to be highly efficient when structuring expressive DL concept spaces (§4.1);

- The development of a novel splitting algorithm for refining numerical concrete domains that leverages the context graph and context-specific interpretations (§4.4, Algorithm 6);

- The development of a versatile, general, memory bounded, top-*k* stochastic beam search algorithm for DL learning that employs our novel refinement operators (§5.1.2, Algorithm 7) [80, 81];

- Various improvements to a utility function (Definition 5.1.5), method of coverage testing (Algorithm 11), and learning (Algorithm 7) based on an analysis of the convexity of performance measures (§5.1.1.2) [81];

- Methods to mitigate redundancy and improperness of refinement with practical strategies around concept normalisation (§5.3.1) and suspension in refinement (§5.3.2);

- The application of DL learning to the real-world problem of protein crystallisation (Chapter 7) [82].

Our implementation OWL-Miner was constructed to verify the performance of the novel methods presented in this thesis. Chapter 6 presented results of our analysis of the performance of OWL-Miner against the state-of-the-art DL learning system called DL-Learner, currently the only other comparable system. We found that OWL-Miner outperformed DL-Learner on several significant benchmark problems. Furthermore, OWL-Miner offers additional modes of learning than DL-Learner by including top-*k* and stochastic beam search for both classification and subgroup discovery with a variety of convex measures including χ^2, WRA, and MCC.

We have been motivated to develop improved methods of DL learning by real-world domains in the life sciences such as protein crystallisation which was discussed in our case study in Chapter 7. Domains which integrate large amounts of RDF data and knowledge with RDFS and OWL ontologies which have a need to derive patterns from the data are ideally suited to the application of our methods, especially where such patterns need to be comprehensible by their users. In the domain of protein crystallisation, OWL classes which are induced from experimental data captured as RDF are helping crystallographers understand more about their scientific domain, and will aid in their experimental discovery process.

8.2 Future Work

8.2.1 Application Areas

8.2.1.1 Life Sciences

As discussed in Chapter 7, we are currently deploying OWL-Miner as a real-time analytical tool for aiding crystallographers in performing scientific experimentation with the development of the X-plorer system. Our experience thus far suggests that OWL-Miner is an ideal technology for aiding domain experts who may not have experience with machine learning or data mining. In particular, the combination of the X-plorer and OWL-Miner systems will enable its users to clearly understand patterns in their data in domain-specific language which can lead to actionable knowledge and measurable gains in efficiency. The domain of protein crystallisation is also interesting from the perspective that the OWL-Miner system is deployed to operate in real-time, as well as in an off-line batch mode. In off-line processing, we aim to incorporate more experimental datasets from laboratories other than C3, such as the Hauptman-Woodward Medical Research Institute's historical crystallisation dataset which can be explored at: http://xtuition.org.

Other domains in which OWL-Miner could be applied in the life sciences are ones which collate rich experimental data and knowledge-based datasets in RDF and OWL for batch-style analysis, such as the Kidney and Urinary Pathway Knowledge Base (KUPKB) [49]. The KUPKB contains a large amount of RDF data representing the results of clinical tests and bioassays in nephrology, with a view to supporting the exploration and analysis of the data to find clinical indicators of various kidney diseases.

The abundance of data being generated and described in the life sciences using RDF and OWL is evidenced by the ever growing number of ontologies hosted by the NCBI BioPortal (http://bioportal.bioontology.org). The BioPortal currently references 524 distinct OWL ontologies covering 7.8 million classes across domains as diverse as agriculture, biological anatomy, genetics and medicine. Tools like OWL-Miner and interfaces like X-plorer can play an important role in analysing knowledge rich data sets in these domains directly, without requiring transformation to other data models or formalisms for analysis with different machine learning tools.

8.2.1.2 Integration with Temporal and Spatial Data

The W3C Spatial Data on the Web Working Group[1] is currently standardising vocabularies for spatial and temporal data for use on the web with RDF and OWL [101]. As the prevalence of such data increases, we anticipate that learning concepts over such data with spatial and temporal relationships will become important. However, deduction with spatial and temporal properties lies outside the scope of traditional DL reasoning algorithms. For example, the Region Connection Calculus (RCC) [79] and Allen's Interval Calculus (AIC) [1] require specialised algorithms for the efficient deduction of relationships and prior exhaustive enumeration is an intractable problem.

As OWL-Miner works by materialising all known inferences up front into a closed-world knowledge base, this approach is not feasible with datasets which rely on inference in RCC, AIC, or similar. For this reason, it would be interesting to investigate how one may incorporate efficient algorithms for computing spatial and temporal relationships between objects, such as those by Renz [83], into the coverage checking methods for learning which we have developed.

8.2.2 Improvements to OWL-Miner

8.2.2.1 Handling Incomplete Data

The methods presented in this thesis rely on the assumption that an OWL knowledgebase $\mathcal{K}$ is *consistent* relative to its closed-world interpretation $\mathcal{J}$ (Definition 3.2.11). As discussed in Section 3.5.2, certain problems may arise from learning from interpretations as per Definition 3.5.1 when $\mathcal{J} \not\models \mathcal{T}$, namely, when the closed-world interpretation is not a model of the TBox. This situation appears to arise when particular assertions of data to the ABox describing examples are *missing* with respect to axioms in the TBox. OWL-Miner does not recognise when a closed-world interpretation used for learning is not a model of the TBox, and therefore may induce concepts which are unsatisfiable with respect to the TBox and its open-world interpretation $\mathcal{I}$ (Example 3.5.4), or induce concepts which appear to solve learning problems relative to $\mathcal{J}$ but which do not perform similarly relative to $\mathcal{I}$ (Example 3.5.6).

Further work is required to understand the impact of learning in DL knowledgebases by closed-world interpretation. As discussed in Section 3.5.2.1, several suggestions for strategies to mitigate these problems may be interpolating missing data

[1]Spatial Data on the Web Working Group: https://www.w3.org/2015/spatial/

with statistical methods, or excluding whole examples or parts thereof. Such methods could be used to ensure that learned concepts are both consistent with background knowledge in a TBox and that their performance measures can be interpreted equally in both closed-world and open-world interpretations. Addressing this problem would ensure that concepts learned by closed-world interpretation in DL knowledge bases could be incorporated back into the knowledge-base they were induced over without causing the knowledge base to become inconsistent.

8.2.2.2 Leveraging Labelled Data in Supervised Learning

As we saw in Section 4.4, our method for learning over concrete domains made use of the labels attributed to examples from which they were reachable by instance chains (Definition 4.2.13). This permitted us to define sensible splitting points when refining numerical range restrictions on the assumption that concepts were to be generated for the purpose of distinguishing examples labelled with one label over all others, such as when solving a machine learning classification problem. A similar approach could be adopted to analyse the distribution of labels amongst instances in local domains over abstract individuals (Definition 4.2.5). If such a distribution could be deduced, a context-specific refinement operator such as $\rho_{\bar{\lambda}}$ could leverage this information to decide whether to refine to certain subexpressions based on the primary label of the examples they are expected to cover. We anticipate that such an approach would further reduce the search space of concepts considered by $\rho_{\bar{\lambda}}$ which ought to further improve the efficiency of the learning methods we have developed.

8.3 Outlook

DL learning has emerged as a powerful method of producing comprehensible descriptions of patterns in RDF/OWL knowledge bases. As the prevalence of RDF and OWL-based data and knowledge on the web continues to increase, DL learning methods which scale to even greater amounts of data and knowledge are increasingly important. Similarly, the need for rich OWL ontologies to describe the abundance of RDF data will increase, and DL learning tools are ideally suited to support their development.

Bibliography

[1] James F. Allen. "Maintaining Knowledge About Temporal Intervals". In: *Commun. ACM* 26.11 (Nov. 1983), pp. 832–843. ISSN: 0001-0782. DOI: 10.1145/182.358434.

[2] Neer Asherie. "Protein crystallization and phase diagrams". In: *Methods (San Diego, Calif.)* 34.3 (Nov. 2004), pp. 266–272. ISSN: 1046-2023. DOI: 10.1016/j.ymeth.2004.03.028.

[3] Martin Atzmueller, Frank Puppe, and Hans-Peter Buscher. "Exploiting background knowledge for knowledge-intensive subgroup discovery". In: *Proceedings of the 19th international joint conference on Artificial intelligence*. IJCAI 2005. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, pp. 647–652.

[4] Franz Baader, Sebastian Brand, and Carsten Lutz. "Pushing the EL envelope". In: *Proceedings of the 19th international joint conference on Artificial intelligence*. IJCAI 2005. 2005, pp. 364–369.

[5] Franz Baader and Ralf Küsters. "Nonstandard Inferences in Description Logics: The Story So Far". In: *Mathematical Problems from Applied Logic I*. Ed. by Dov M. Gabbay, Sergei S. Goncharov, and Michael Zakharyaschev. Vol. 4. New York: Springer-Verlag, 2006, pp. 1–75. ISBN: 0-387-28688-8.

[6] Franz Baader et al., eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press, 2003. ISBN: 0-521-78176-0.

[7] Liviu Badea and Shan-Hwei Nienhuys-Cheng. "A Refinement Operator for Description Logics". In: *Proceedings of the 10th International Conference on Inductive Logic Programming*. ILP '00. ACM ID: 742934. Springer-Verlag, 2000, pp. 40–59. ISBN: 3-540-67795-X.

[8] Christopher J. O. Baker and Kei-Hoi Cheung, eds. *Semantic web : revolutionizing knowledge discovery in the life sciences*. eng. Springer, 2007. ISBN: 0-387-48436-1,978-0-387-48436-5.

[9] Helen M Berman et al. "The protein structure initiative structural genomics knowledgebase". In: *Nucleic Acids Research* 37.Database issue (Jan. 2009), pp. D365–368. ISSN: 1362-4962. DOI: 10.1093/nar/gkn790.

[10] T. Berners-Lee and J. Hendler. "Scientific publishing on the semantic web". In: *Nature* 410 (2001), pp. 1023–1024.

[11] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web: Scientific American". In: *Scientific American* (May 2001).

[12] F. C. Bernstein et al. "The Protein Data Bank: a computer-based archival file for macromolecular structures". In: *Journal of Molecular Biology* 112.3 (May 1977), pp. 535–542. ISSN: 0022-2836.

[13] C. Bizer, T. Heath, and T. Berners-Lee. "Linked data - The story so far". In: *International Journal on Semantic Web and Information Systems* 5.3 (2009), pp. 1–22.

[14] Alex Borgida. *On the Relative Expressiveness of Description Logics and Predicate Logics*. 1996.

[15] Robert Cattral, Franz Oppacher, and Dwight Deugo. "Evolutionary data mining with automatic rule generalization". In: *Recent Advances in Computers, Computing and Communications*. 2002, pp. 296–300.

[16] Edited by Naomi E. Chayen. *Protein crystallization strategies for structural genomics*. 1st ed. International University Line, July 2007. ISBN: 0-9720774-3-X.

[17] Naomi E Chayen and Emmanuel Saridakis. "Protein crystallization: from purified protein to diffraction-quality crystal". en. In: *Nature Methods* 5.2 (Jan. 2008), pp. 147–153. ISSN: 1548-7091. DOI: 10.1038/nmeth.f.203.

[18] William W Cohen and Haym Hirsh. "Learning the CLASSIC Description Logic: Theoretical and Experimental Results". In: *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR'94)* (1994), pp. 121–133.

[19] Fabio Gagliardi Cozman and Rodrigo Bellizia Polastro. "Loopy Propagation in a Probabilistic Description Logic". In: *Proceedings of the 2nd international conference on Scalable Uncertainty Management*. SUM '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 120–133. ISBN: 978-3-540-87992-3. DOI: 10.1007/978-3-540-87993-0_11.

[20] Bernardo Cuenca Grau et al. "Incremental Classification of Description Logics Ontologies". In: *Journal of Automated Reasoning* 44.4 (2010), pp. 337–369. ISSN: 0168-7433. DOI: 10.1007/s10817-009-9159-0.

[21] Christian Cumbaa and Igor Jurisica. "Automatic classification and pattern discovery in high-throughput protein crystallization trials". In: *Journal of Structural and Functional Genomics* 6.2-3 (2005), pp. 195–202. ISSN: 1345-711X. DOI: 10.1007/s10969-005-5243-9.

[22] Chad Cumby and Dan Roth. "On Kernel Methods for Relational Learning". English (US). In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*. 2003.

[23] Luc De Raedt. *Logical and Relational Learning*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 978-3-642-05748-9.

[24] Asim Kumar Debnath et al. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". In: *Journal of Medicinal Chemistry* 34.2 (Feb. 1991), pp. 786–797. ISSN: 0022-2623. DOI: 10.1021/jm00106a046.

[25] Kirill Degtyarenko et al. "ChEBI: a database and ontology for chemical entities of biological interest". In: *Nucleic Acids Research* 36.Database issue (Jan. 2008), pp. D344–350. ISSN: 1362-4962. DOI: 10.1093/nar/gkm791.

[26] Lawrence J DeLucas et al. "Efficient protein crystallization". In: *Journal of Structural Biology* 142.1 (Apr. 2003), pp. 188–206. ISSN: 1047-8477.

[27] Inês de Castro Dutra et al. "An Empirical Evaluation of Bagging in Inductive Logic Programming". en. In: *Inductive Logic Programming*. Ed. by Stan Matwin and Claude Sammut. Lecture Notes in Computer Science 2583. DOI: 10.1007/3-540-36468-4_4. Springer Berlin Heidelberg, July 2002, pp. 48–65. ISBN: 978-3-540-00567-4 978-3-540-36468-9.

[28] Howard Einspahr and Manfred S. Weiss. "Call for a crystallization ontology". In: *Acta Crystallographica Section F Structural Biology and Crystallization Communications* 68.3 (Feb. 2012), pp. 252–252. ISSN: 1744-3091. DOI: 10.1107/S174430911200680X.

[29] Floriana Esposito et al. "A counterfactual-based learning algorithm for ALC description logic". In: *Proceedings of the 9th conference on Advances in Artificial Intelligence*. AI*IA'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 406–417. ISBN: 3-540-29041-9 978-3-540-29041-4. DOI: 10.1007/11558590_41.

[30] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. "DL-FOIL Concept Learning in Description Logics". In: *Inductive Logic Programming*. Ed. by Filip Železný and Nada Lavrač. Vol. 5194. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 107–121. ISBN: 978-3-540-85927-7 978-3-540-85928-4.

[31] Robert G. Farr, Alexander L. Perryman, and Cleopas T. Samudzi. "Re-clustering the database for crystallization of macromolecules". In: *Journal of Crystal Growth* 183.4 (Feb. 1998), pp. 653–668. ISSN: 0022-0248. DOI: 16/S0022-0248(97)00492-2.

[32] Johannes Fürnkranz and Peter A. Flach. "An Analysis of Rule Evaluation Metrics". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. Ed. by Tom E. Fawcett and N. Mishra. Washington, DC: AAAI Press, 2003, pp. 202–209.

[33] Johannes Fürnkranz and Peter A. Flach. "ROC 'n' Rule Learning—Towards a Better Understanding of Covering Algorithms". en. In: *Machine Learning* 58.1 (Jan. 2005), pp. 39–77. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-005-5011-x.

[34] Thomas Gärtner et al. "Multi-Instance Kernels". In: *In Proc. 19th International Conf. on Machine Learning*. Morgan Kaufmann, 2002, pp. 179–186.

[35] Birte Glimm et al. "Optimising Ontology Classification". en. In: *The Semantic Web – ISWC 2010*. Ed. by Peter F. Patel-Schneider et al. Lecture Notes in Computer Science 6496. DOI: 10.1007/978-3-642-17746-0_15. Springer Berlin Heidelberg, Nov. 2010, pp. 225–240. ISBN: 978-3-642-17745-3 978-3-642-17746-0.

[36] Vanathi Gopalakrishnan et al. "Machine-learning techniques for macromolecular crystallization data". In: *Acta Crystallographica. Section D, Biological Crystallography* 60.Pt 10 (Oct. 2004), pp. 1705–1716. ISSN: 0907-4449. DOI: 10.1107/S090744490401683X.

[37] Georg Gottlob and Christos Papadimitriou. "On the complexity of single-rule datalog queries". In: *Information and Computation* 183.1 (May 2003), pp. 104–122. ISSN: 0890-5401. DOI: 10.1016/S0890-5401(03)00012-9.

[38] Janna Hastings et al. "Structure-based classification and ontology in chemistry". en. In: *Journal of Cheminformatics* 4.1 (Apr. 2012), p. 8. ISSN: 1758-2946. DOI: 10.1186/1758-2946-4-8.

[39] D Hennessy et al. "Induction of rules for biological macromolecule crystallization". In: *Proceedings of the International Conference on Intelligent Systems for Molecular Biology; ISMB. International Conference on Intelligent Systems for Molecular Biology* 2 (1994), pp. 179–187. ISSN: 1553-0833.

[40] Francisco Herrera et al. "An overview on subgroup discovery: foundations and applications". en. In: *Knowledge and Information Systems* 29.3 (Dec. 2011), pp. 495–525. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-010-0356-2.

[41] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for OWL ontologies". In: *Semantic Web* 2.1 (Jan. 2011), pp. 11–21. DOI: 10.3233/SW-2011-0025.

[42] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. "The Even More Irresistible SROIQ". In: *Proc of the 10th Int Conf on Principles of Knowledge Representation and Reasoning KR 2006* (2006), pp. 1–36.

[43] Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. "An algorithm based on counterfactuals for concept learning in the Semantic Web". In: *Applied Intelligence* 26.2 (Apr. 2007), pp. 139–159. ISSN: 0924-669X. DOI: 10.1007/s10489-006-0011-5.

[44] Ning Jiang and Simon Colton. "Boosting Descriptive ILP for Predictive Learning in Bioinformatics". en. In: *Inductive Logic Programming*. Ed. by Stephen Muggleton, Ramon Otero, and Alireza Tamaddoni-Nezhad. Lecture Notes in Computer Science 4455. DOI: 10.1007/978-3-540-73847-3_28. Springer Berlin Heidelberg, Aug. 2006, pp. 275–289. ISBN: 978-3-540-73846-6 978-3-540-73847-3.

[45] I. Jurisica et al. "Intelligent decision support for protein crystal growth". In: *IBM Systems Journal* 40.2 (2001), pp. 394–409. ISSN: 0018-8670. DOI: 10.1147/sj.402.0394.

[46] Yevgeny Kazakov. "RIQ and SROIQ are harder than SHOIQ". In: *In Proc. KR'08*. 2008.

[47] Yevgeny Kazakov. "$\mathcal{SRIQ}$ and $\mathcal{SROIQ}$ are Harder than $\mathcal{SHOIQ}$". In: *Proceedings of the 21st International Workshop on Description Logics*. Ed. by Franz Baader, Carsten Lutz, and Boris Motik. Vol. 353. http://ceur-ws.org: CEUR Workshop Proceedings, May 2008.

[48] R D King, A Srinivasan, and L Dehaspe. “Warmr: a data mining tool for chemical data”. In: *Journal of Computer-Aided Molecular Design* 15.2 (Feb. 2001), pp. 173–181. ISSN: 0920-654X.

[49] Julie Klein et al. “The KUPKB: a novel Web application to access multiomics data on kidney disease”. eng. In: *FASEB journal: official publication of the Federation of American Societies for Experimental Biology* 26.5 (May 2012), pp. 2145–2153. ISSN: 1530-6860. DOI: 10.1096/fj.11-194381.

[50] Stefan Kramer and Luc De Raedt. “Feature Construction with Version Spaces for Biochemical Applications”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 258–265. ISBN: 978-1-55860-778-1.

[51] Mark-A. Krogel et al. “Comparative Evaluation of Approaches to Propositionalization”. en. In: *Inductive Logic Programming*. Ed. by Tamás Horváth and Akihiro Yamamoto. Lecture Notes in Computer Science 2835. DOI: 10.1007/978-3-540-39917-9_14. Springer Berlin Heidelberg, Sept. 2003, pp. 197–214. ISBN: 978-3-540-20144-1 978-3-540-39917-9.

[52] Werner Kühlbrandt. “Cryo-EM enters a new era”. en. In: *eLife* 3 (Aug. 2014), e03678. ISSN: 2050-084X. DOI: 10.7554/eLife.03678.

[53] Niels Landwehr, Kristian Kersting, and Luc De Raedt. “nFOIL: Integrating NaïVe Bayes and FOIL”. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*. AAAI’05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 795–800. ISBN: 978-1-57735-236-5.

[54] Nada Lavrač, Peter Flach, and Blaz Zupan. “Rule Evaluation Measures: A Unifying View”. en. In: *Inductive Logic Programming*. Ed. by Sašo Džeroski and Peter Flach. Lecture Notes in Computer Science 1634. DOI: 10.1007/3-540-48751-4_17. Springer Berlin Heidelberg, June 1999, pp. 174–185. ISBN: 978-3-540-66109-2 978-3-540-48751-7.

[55] Agnieszka Ławrynowicz and Jedrzej Potoniec. “Fr-ONT: An Algorithm for Frequent Concept Mining with Formal Ontologies”. In: *Foundations of Intelligent Systems*. Ed. by Marzena Kryszkiewicz et al. Lecture Notes in Computer Science 6804. Springer Berlin Heidelberg, Jan. 2011, pp. 428–437. ISBN: 978-3-642-21915-3 978-3-642-21916-0.

[56] Jens Lehmann. “DL-Learner: Learning Concepts in Description Logics”. In: *Journal of Machine Learning Research (JMLR)* 10 (2009), pp. 2639–2642.

[57] Jens Lehmann and Christoph Haase. "Ideal downward refinement in the EL description logic". In: *Proceedings of the 19th international conference on Inductive logic programming*. ILP'09. ACM ID: 1893546. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 73–87. ISBN: 3-642-13839-X 978-3-642-13839-3.

[58] Jens Lehmann and Pascal Hitzler. "Concept learning in description logics using refinement operators". In: *Machine Learning* 78.1-2 (Sept. 2009), pp. 203–250. ISSN: 0885-6125. DOI: 10.1007/s10994-009-5146-2.

[59] Jens Lehmann and Pascal Hitzler. "Foundations of Refinement Operators for Description Logics". In: *Inductive Logic Programming*. Ed. by Hendrik Blockeel et al. Lecture Notes in Computer Science 4894. Springer Berlin Heidelberg, Jan. 2008, pp. 161–174. ISBN: 978-3-540-78468-5 978-3-540-78469-2.

[60] Jens Lehmann et al. "Class expression learning for ontology engineering". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 9.1 (Mar. 2011), pp. 71–81. ISSN: 1570-8268. DOI: 10.1016/j.websem.2011.01.001.

[61] Huma Lodhi and Stephen Muggleton. "Is mutagenesis still challenging?" In: *In: Proceedings of the 15th International Conference on Inductive Logic Programming, ILP 2005, Late-Breaking Papers. (2005) 35–40*. 2005, pp. 35–40.

[62] Pierre Mahé et al. "Graph Kernels for Molecular Structure-Activity Relationship Analysis with Support Vector Machines". In: *Journal of Chemical Information and Modeling* 45.4 (July 2005), pp. 939–951. ISSN: 1549-9596. DOI: 10.1021/ci050039t.

[63] Alexander McPherson. *Crystallization of Biological Macromolecules*. New York: Cold Spring Harbor Laboratory Press, 1999. ISBN: 0-87969-617-6.

[64] R Michalski and J Larson. "Inductive inference of VL decision rules". In: *Proceedings of the Workshop in Pattern-Directed Inference Systems (Published in SIGART Newsletter ACM)*. 1977, pp. 38–44.

[65] Shinichi Morishita and Jun Sese. "Transversing Itemset Lattices with Statistical Metric Pruning". In: *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '00. New York, NY, USA: ACM, 2000, pp. 226–236. ISBN: 978-1-58113-214-4. DOI: 10.1145/335168.335226.

[66] Boris Motik and Ian Horrocks. "OWL Datatypes: Design and Implementation". In: *Proceedings of the 7th International Conference on The Semantic Web*. ISWC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 307–322. ISBN: 978-3-540-88563-4. DOI: 10.1007/978-3-540-88564-1_20.

[67] Stephen Muggleton and Cao Feng. "Efficient Induction Of Logic Programs". In: *New Generation Computing*. Academic Press, 1990.

[68] Stephen Muggleton and Luc De Raedt. "Inductive Logic Programming: Theory and Methods". In: *Journal of Logic Programming* 19.20 (1994), pp. 629–679.

[69] Raymond M Nagel, Joseph R Luft, and Edward H Snell. "AutoSherlock: a program for effective crystallization data analysis". In: *Journal of Applied Crystallography* 41.Pt 6 (Dec. 2008), pp. 1173–1176. ISSN: 0021-8898. DOI: 10.1107/S0021889808028938.

[70] Darren A Natale et al. "The Protein Ontology: a structured representation of protein forms and complexes". In: *Nucleic Acids Research* 39.Database issue (Jan. 2011), pp. D539–545. ISSN: 1362-4962. DOI: 10.1093/nar/gkq907.

[71] Janet Newman et al. "On the need for an international effort to capture, share and use crystallization screening data". In: *Acta Crystallographica Section F Structural Biology and Crystallization Communications* 68.3 (Feb. 2012), pp. 253–258. ISSN: 1744-3091. DOI: 10.1107/S1744309112002618.

[72] Kee Siong Ng. "Learning Comprehensible Theories from Structured Data". PhD thesis. Research School of Information Sciences, Engineering, and The Australian National University, Oct. 2005.

[73] José Eduardo Ochoa-Luna, Kate Revoredo, and Fábio Gagliardi Cozman. "Learning probabilistic description logics: a framework and algorithms". In: *Proceedings of the 10th Mexican international conference on Advances in Artificial Intelligence - Volume Part I*. MICAI'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 28–39. ISBN: 978-3-642-25323-2. DOI: 10.1007/978-3-642-25324-9_3.

[74] Rebecca Page and Raymond C Stevens. "Crystallization data mining in structural genomics: using positive and negative results to optimize protein crystallization screens". In: *Methods (San Diego, Calif.)* 34.3 (Nov. 2004), pp. 373–389. ISSN: 1046-2023. DOI: 10.1016/j.ymeth.2004.03.026.

[75] Thomas S Peat, Jon A Christopher, and Janet Newman. "Tapping the Protein Data Bank for crystallization information". In: *Acta Crystallographica. Section D, Biological Crystallography* 61.Pt 12 (Dec. 2005), pp. 1662–1669. ISSN: 0907-4449. DOI: 10.1107/S0907444905033202.

[76] David Martin Ward Powers. *Evaluation: from precision, recall and F-factor to ROC, informedness, markedness and correlation*. SIE-07-001. Flinders University, Adelaide, Australia, 2007.

[77] J. R. Quinlan. "Boosting first-order learning". en. In: *Algorithmic Learning Theory*. Ed. by Setsuo Arikawa and Arun K. Sharma. Lecture Notes in Computer Science 1160. DOI: 10.1007/3-540-61863-5_42. Springer Berlin Heidelberg, Oct. 1996, pp. 143–155. ISBN: 978-3-540-61863-8 978-3-540-70719-6.

[78] J. R. Quinlan. "Learning logical definitions from relations". In: *Machine Learning* 5 (1990), pp. 239–266.

[79] David A. Randell, Zhan Cui, and Anthony G. Cohn. "A Spatial Logic based on Regions and Connection". In: *Proceedings 3rd International Conference on Knowledge Representation and Reasoning*. 1992.

[80] David Ratcliffe and Kerry Taylor. "Closed-World Concept Induction for Learning in OWL Knowledge Bases". In: *Knowledge Engineering and Knowledge Management*. Ed. by Krzysztof Janowicz et al. Lecture Notes in Computer Science 8876. DOI: 10.1007/978-3-319-13704-9_33. Springer International Publishing, Nov. 2014, pp. 429–440. ISBN: 978-3-319-13703-2 978-3-319-13704-9.

[81] David Ratcliffe and Kerry Taylor. "Refinement-Based OWL Class Induction with Convex Measures". In: *Semantic Technology - 7th Joint International Conference, JIST 2017, Gold Coast, QLD, Australia, November 10-12, 2017, Proceedings*. 2017, pp. 49–65. DOI: 10.1007/978-3-319-70682-5.

[82] David Ratcliffe, Kerry Taylor, and Janet Newman. "Ontology-based machine learning for protein crystallisation". In: *Australasian Ontology Workshop (AOW 2011)*. Vol. 132. Perth, Australia: CRPIT, 2011.

[83] Jochen Renz. "Qualitative Spatial and Temporal Reasoning: Efficient Algorithms for Everyone". In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 526–531.

[84] J. Rissanen. "Modeling by shortest data description". In: *Automatica* 14.5 (Sept. 1978), pp. 465–471. ISSN: 0005-1098. DOI: 10.1016/0005-1098(78)90005-5.

[85] Walter Rudin. *Principles of mathematical analysis / Walter Rudin*. International series in pure and applied mathematics. Includes index. Bibliography: p. [335]-336. New York: McGraw-Hill, 1976. ISBN: 978-0-07-054235-8.

[86] Bernhard Rupp and Junwen Wang. "Predictive models for protein crystallization". In: *Methods (San Diego, Calif.)* 34.3 (Nov. 2004), pp. 390–407. ISSN: 1046-2023. DOI: 10.1016/j.ymeth.2004.03.031.

[87] Jeffrey Curtis Schlimmer. "Concept Acquisition Through Representational Adjustment". AAI8724747. PhD thesis. University of California, Irvine, 1987.

[88] Stefan Schulz, Kornél Markó, and Boontawee Suntisrivaraporn. "Formal representation of complex SNOMED CT expressions". In: *BMC Medical Informatics and Decision Making* 8 Suppl 1 (2008), S9. ISSN: 1472-6947. DOI: 10.1186/1472-6947-8-S1-S9.

[89] Michele Sebag and Celine Rouveirol. "Tractable Induction and Classification in First Order Logic via Stochastic Matching". In: *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence - Volume 2*. IJCAI'97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 888–893.

[90] N. Shadbolt, W. Hall, and T. Berners-Lee. "The Semantic Web Revisited". English. In: *IEEE Intelligent Systems* 21.3 (Feb. 2006), pp. 96–101. ISSN: 1541-1672. DOI: 10.1109/MIS.2006.62.

[91] 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)*. 2008.

[92] Evren Sirin et al. "Pellet: A practical OWL-DL reasoner". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.2 (June 2007), pp. 51–53. ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.03.004.

[93] Edward H Snell et al. "Establishing a training set through the visual analysis of crystallization trials. Part I: approximately 150,000 images". In: *Acta Crystallographica. Section D, Biological Crystallography* 64.Pt 11 (Nov. 2008), pp. 1123–1130. ISSN: 1399-0047. DOI: 10.1107/S0907444908028047.

[94] Edward H. Snell et al. "The application and use of chemical space mapping to interpret crystallization screening results". In: *Acta Crystallographica Section D Biological Crystallography* 64.12 (Nov. 2008), pp. 1240–1249. ISSN: 0907-4449. DOI: 10.1107/S0907444908032411.

[95] Ashwin Srinivasan et al. "Theories for mutagenicity: a study in first-order and feature-based induction". In: *Artificial Intelligence* 85.1 (Aug. 1996), pp. 277–299. ISSN: 0004-3702. DOI: 10.1016/0004-3702(95)00122-0.

[96] A. Srinivasan et al. "Carcinogenesis predictions using ILP". en. In: *Inductive Logic Programming*. Ed. by Nada Lavrač and Sašo Džeroski. Lecture Notes in Computer Science 1297. DOI: 10.1007/3540635149_56. Springer Berlin Heidelberg, Sept. 1997, pp. 273–287. ISBN: 978-3-540-63514-7 978-3-540-69587-5.

[97] Gerd Stumme, Andreas Hotho, and Bettina Berendt. "Semantic Web Mining". In: *Web Semant.* 4.2 (June 2006), pp. 124–143. ISSN: 1570-8268. DOI: 10.1016/j.websem.2006.02.001.

[98] Alireza Tamaddoni-Nezhad and Stephen Muggleton. "Stochastic Refinement". en. In: *Inductive Logic Programming*. Ed. by Paolo Frasconi and Francesca A. Lisi. Lecture Notes in Computer Science 6489. DOI: 10.1007/978-3-642-21295-6_26. Springer Berlin Heidelberg, June 2010, pp. 222–237. ISBN: 978-3-642-21294-9 978-3-642-21295-6.

[99] Jiao Tao. "Integrity Constraints for the Semantic Web: An OWL2-DL Extension". AAI3530046. PhD thesis. Troy, NY, USA: Rensselaer Polytechnic Institute, 2012.

[100] Jiao Tao et al. "Integrity constraints in OWL". In: *AAAI2010: Proceedings of the AAAI Conference on Artificial Intelligence*. 2010.

[101] K. Taylor and E. Parsons. "Where Is Everywhere: Bringing Location to the Web". In: *IEEE Internet Computing* 19.2 (Mar. 2015), pp. 83–87. ISSN: 1089-7801. DOI: 10.1109/MIC.2015.50.

[102] Thomas Taylor. *Twelve Edible Mushrooms of the United States, USDA*. 1894.

[103] "The Gene Ontology project in 2008". In: *Nucleic Acids Research* 36.Database issue (Jan. 2008), pp. D440–D444. ISSN: 0305-1048. DOI: 10.1093/nar/gkm883.

[104] An C. Tran et al. "An approach to numeric refinement in description logic learning for learning activities duration in smart homes". In: *AAAI-13 Workshop on Space, Time, and Ambient Intelligence*. Bellevue, Washington, July 2013.

[105] Michael Tung and D Travis Gallagher. "The Biomolecular Crystallization Database Version 4: expanded content and new features". In: *Acta Crystallographica. Section D, Biological Crystallography* 65.Pt 1 (Jan. 2009), pp. 18–23. ISSN: 1399-0047. DOI: 10.1107/S0907444908035440.

[106] Anni-Yasmin Turhan. “Pushing the SONIC border – SONIC 1.0”. In: *Proc. of Fifth International Workshop on First-Order Theorem Proving (FTP 2005). Technical Report University of Koblenz, 2005. 2007/TONES – March 31, 2007 55/55 TONES-D15 – v.1.1* (2007).

[107] Pascal Vallotton et al. “DroplIT , an improved image analysis method for droplet identification in high-throughput crystallization trials”. In: *Journal of Applied Crystallography* 43.6 (2010), pp. 1548–1552. DOI: 10.1107/S0021889810040963.

[108] Christopher G. Walker, James Foadi, and Julie Wilson. “Classification of protein crystallization images using Fourier descriptors”. In: *Journal of Applied Crystallography* 40.3 (June 2007), pp. 418–426. ISSN: 0021-8898. DOI: 10.1107/S0021889807011156.

[109] Stefan Wrobel. “An Algorithm for Multi-relational Discovery of Subgroups”. In: *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery*. PKDD ’97. London, UK, UK: Springer-Verlag, 1997, pp. 78–87. ISBN: 3-540-63223-9.

[110] Filip Železny, Ashwin Srinivasan, and David Page. “Lattice-search Runtime Distributions May Be Heavy-tailed”. In: *Proceedings of the 12th International Conference on Inductive Logic Programming*. ILP’02. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 333–345. ISBN: 978-3-540-00567-4.

[111] Albrecht Zimmermann and Luc De Raedt. “Cluster-grouping: from subgroup discovery to clustering”. en. In: *Machine Learning* 77.1 (June 2009), pp. 125–159. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-009-5121-y.